The Dissertation Committee for Chad Allan Baker
certifies that this is the approved version of the following dissertation:

Simulation, Design, and Experimental Characterization of Catalytic and Thermoelectric Systems for Removing Emissions and Recovering Waste Energy from Engine Exhaust

Committee:

Matthew J. Hall, Supervisor

Li Shi, Co-Supervisor

Ronald D. Matthews

Ofodike A. Ezekoye

Laxminarayan L. Raja

Simulation, Design, and Experimental Characterization of Catalytic and Thermoelectric Systems for Removing Emissions and Recovering Waste Energy from Engine Exhaust

by

Chad Allan Baker, B.S.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

The University of Texas at Austin

December 2012

Acknowledgments

I owe a tremendous amount of gratitude to my advisor, Dr. Matthew Hall, for his exceptional mentoring and guidance to help me reach this goal. Dr. Hall was easily accessible when I needed help, willing to correct me if I got too far on the wrong track, and most important of all, giving me enough room to make my own mistakes and learn from them. His feedback was direct and highly constructive, be it praise or criticism. In addition to the obvious technical expertise he taught me, by acting as an ideal example, he taught me many valuable lessons about how to write well, how to interact with fellow engineers, how to interact with machinists, and how to accept and benefit from peers' criticisms. As much as I hope to match Dr. Hall's technical abilities in my own career, I hope even more that I can emulate his straightforward, honest, supportive manner of dealing with students, subordinates, and coworkers.

Dr. Li Shi has also been a great help in this process. It was easy to see that he cares a great deal about the knowledge and experience that he provides for his graduate students and students in his classes. He teaches for all of the right reasons, and his passion for his career is readily apparent. Dr. Shi aggressively introduced me to potential employers every time we attended a conference together, and because of his enthusiasm for helping his students, I know he will have a lasting impact on the engineering profession.

I wish to thank my wife, Dr. Kerrie Gath, for keeping me sane through all the intellectual struggles involved with earning a doctorate. She is my personal compass, keeping me on the best possible path through life, even when there are many otherwise overwhelming obstacles. Her steadfast support was critical to my success.

I'm thankful that she's on my team.

I wish to thank my parents, Allan and Debbie Baker, for their constant support and all of the life lessons they provided me with while I was growing up. I am appreciative of my brother, Kevin Baker, for providing friendship and support. I have learned much from Kevin, and I always strive to emulate his manner of dealing with people from many different backgrounds.

Many people in my extended family, whether they know it or not, have offered encouragement that helped me along. I owe my family a great deal of gratitude for their support. I would especially like to thank my grandmother, Jean Baker, because her encyclopedias and field guides were my first introduction to science, and without these books, my life may have turned out completely different, almost certainly not as good as it has.

I owe thanks to Dr. DK Ezekoye for listening to so many of my wild ideas for modeling various physical phenomena. He has also helped tremendously with teaching me how to write and explain ideas in a pedagogically coherent manner. I wish to thank Dr. Alexandre da Silva for repeatedly emphasizing the idea that CFD code is not the starting point for solving a complicated problem. This concept has helped me to develop my standard method for solving brand new problems: first principles analysis, scaling analysis, literature review of similar concepts, analytical solution if possible, and CFD or experiment if appropriate. I wish to thank Dr. Phil Schmidt for helping me to stay excited about science and engineering, even when engineering seemed like too difficult of a profession.

I greatly appreciate the contributions of my committee members: Matt Hall, Li Shi, Ron Matthews, DK Ezekoye, and Lax Raja. All of my committee members, either through lengthy discussions or the classroom, have provided me with essential

knowledge and skills responsible for generating the content of this document.

In my life before graduate school, there were numerous folks who provided guidance that has gotten me this far. Among these are Keith Nelson, Richard Davis, Gene Wenzel, Kent Thele, Obdulia Ley, Jerald Caton, Tim Jacobs, Luis san Andrés, and Lynne Connel.

Simulation, Design, and Experimental Characterization of Catalytic and Thermoelectric Systems for Removing Emissions and Recovering Waste Energy from Engine Exhaust

Chad Allan Baker, Ph.D.
The University of Texas at Austin, 2012

Supervisors: Matthew J. Hall
Li Shi

An analytical transport/reaction model was developed to simulate the catalytic performance of ZnO nanowires as a catalyst support. ZnO nanowires were chosen because they have easily characterized, controllable features and a spatially uniform morphology. The analytical model couples convection in the catalyst flow channel with reaction and diffusion in the porous substrate material; it was developed to show that a simple analytical model with physics-based mass transport and empirical kinetics can be used to capture the essential physics involved in catalytic conversion of hydrocarbons. The model was effective at predicting species conversion efficiency over a range of temperature and flow rate. The model clarifies the relationship between advection, bulk diffusion, pore diffusion, and kinetics. The model was used to optimize the geometry of the experimental catalyst for which it predicted that maximum species conversion density for fixed catalyst surface occurred at a channel height of 520 μm.

A modeling study of thermoelectric (TE) vehicle waste heat recovery was conducted based on abundant and inexpensive $Mg_2Si_{0.5}Sn_{0.5}$ and $MnSi_{1.75}$ TE materials

with consideration of performance at the system and TE device levels. The modeling study identified a critical TE design space of fill fraction, leg length, n-/p-type leg area ratio, and current; these parameters needed to be optimized simultaneously for positive TE power output. The TE power output was sensitive to this design space, and the optimal design point was sensitive to engine operating conditions. The maximum net TE power for a 29.5 L strip fin heat exchanger with an 800 K exhaust flow at 7.9 kg/min was 2.25 kW. This work also includes two generations of TE waste heat recovery systems that were built and tested in the exhaust system of a Cummins 6.7 L turbo Diesel engine. The first generation was a small scale heat exchanger intended for concept validation, and the second generation was a full scale heat exchanger that used the entire exhaust flow at high speed and torque. The second generation heat exchanger showed that the model could accurately predict heat transfer, and the maximum experimental heat transfer rate was 15.3 kW for exhaust flow at 7.0 kg/min and 740 K.

Table of Contents

List of Tables

List of Figures

Preface

Explanation of Two Topics

This dissertation is divided into two similar topics, both of which have the aim of reducing the amount of pollutants per mile driven emitted from the tailpipe of an automobile. The first of these topics deals with direct catalytic control of emission concentration in the tail pipe, and the second deals with exhaust gas waste heat recovery which enables more efficient vehicle operation, thereby reducing the mass of exhaust emitted by the vehicle.

The two topics are both covered because the work in the catalyst project came to a logical and fiscal conclusion earlier than expected. To provide more substance and content, the thermoelectric project was deemed sufficiently similar to the first project that it made sense to include both projects in this body of work.

In addition, the key intellectual contributions for both projects were considered to be about equal in importance, and both topics were related in terms of the fundamental heat and mass transport considerations. This document will first discuss the catalyst project then the thermoelectric project.

General Motivation

Both of these projects have the effect of reducing the amount of pollution emitted by internal combustion engine powered vehicles by either directly controlling concentration of emissions in the tailpipe via catalysis or reducing total emissions via reduced fuel consumption. The Environmental Protection Agency (EPA) is the government entity that regulates automotive tailpipe emissions, and emissions standards

set forth by the EPA continue to become progressively more stringent. The standards are written in terms of grams of pollutant per mile driven so efficiency as well as pollutant concentration are important in meeting the EPA standards. This is an effective means of regulating pollution because the total amount of pollution emitted by vehicles is a function of both the mass of exhaust produced by vehicles (which decreases with increasing efficiency) and the concentration of the criteria pollutants in the exhaust [1, 2].

In order to avoid taking for granted the notion that regulating air pollution is beneficial, a brief overview of the justifications for such regulation will be presented next. Air pollution regulation has three goals: minimizing environmental damage, minimizing negative health effects, and reducing ambient pollution concentrations such that the cost of the economic externalities created by air pollution is less than the net economic benefit obtained by reducing the harmful effects of air pollution through regulation. Numerous studies have shown that automotive pollutants and the secondary hazardous chemicals they form in the atmosphere, collectively known as smog, cause a great deal of harm to the environment and human health. Acid rain, damage to plants, increased rates of allergies and asthma, respiratory damage in otherwise healthy individuals, increased rates of cancer, and even premature deaths can be attributed either directly or at least indirectly to automotive air pollution [3, 4, 5, 6, 7, 8]. The EPA has also estimated that the economic benefits of regulation outweigh the cost of regulation, in the form of enforcement cost and increased manufacturing expense, by a factor of thirty to one, with an unspecified confidence interval ranging from three to one up to ninety to one [9].

Part I

Analytical Model and Experimental Comparison to ZnO Nanowire Substrates for Transport/Reaction in the Channels of a Hydrocarbon Oxidation Catalyst

Chapter 1

Introduction and Background

1.1 Catalyst Specific Motivation

The cost of catalyst metals is continuously rising as demand increases due, in part, to tightening combustion emissions regulations across the globe. Because of this, there is great incentive to use catalyst metals more effectively for the remediation of combustion emissions. A typical automotive catalyst consists of a ceramic honeycomb monolith, typically cylindrical in shape. For a mid-size passenger car, the catalyst is roughly half a meter in length and about 15 cm in diameter. The honeycomb structure consists of an array of square channels that are roughly 1 mm by 1 mm that run the length of the catalyst monolith. The exhaust flows through these honeycomb channels, and a special porous media coating on the walls of the channels contains the catalyst metals that are involved in reducing the concentration of emissions. This special porous coating is referred to as a wash-coat, and the desired characteristics are high surface area per unity mass, i.e. specific surface area; minimal impact on gaseous molecular diffusion; high thermal stability; and high chemical stability. An SEM image showing an example of a honeycomb array is shown in Figure 1.1.

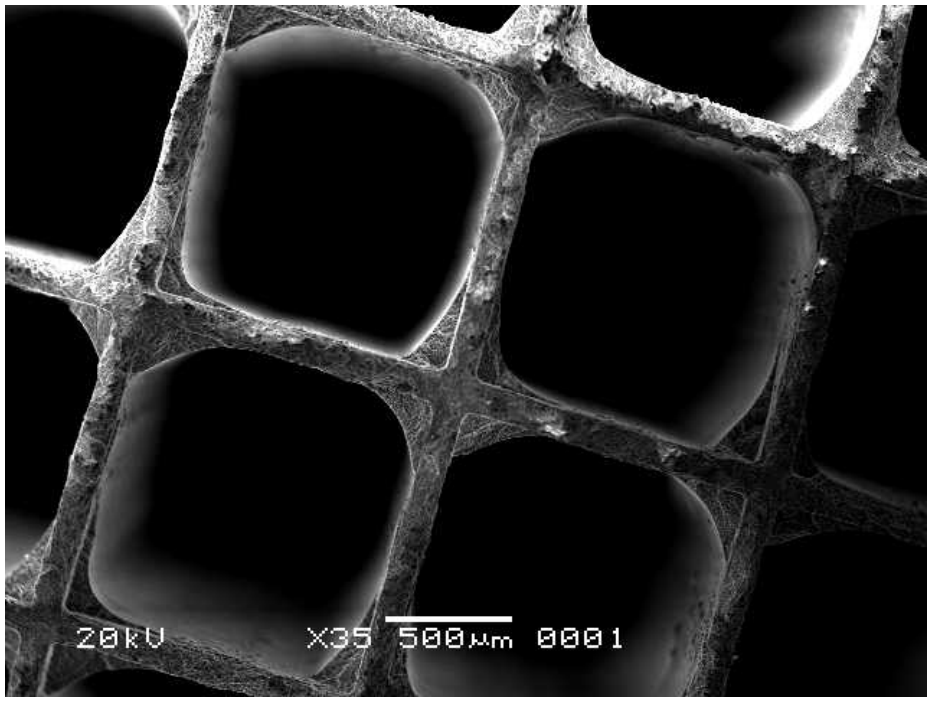


Figure 1.1: SEM image of catalyst honeycomb monolith seen from the end. Wash-coat material is visible in the corners of the channels where the wash-coat is thicker. Image is facing in the stream-wise direction.

This work consists of modeling and experimentation to investigate the performance of a platinum/palladium combustion emissions catalyst that uses ZnO nanowires as a support medium. Comparisons are made among surfaces consisting of: ZnO nanowires grown on a flat plate Si wafer, a flat plate cordierite wafer, and a flat plate γ-alumina-coated cordierite wafer, all sputter coated with Pt/Pd. The experimental results were modeled with an analytical species transport model that couples convection in the catalyst channel with diffusion and reaction in the catalyst substrate media.

1.2 Literature Review

The motivation for the experimental part of this research was to study the effect of using a catalyst support material with a very different surface morphology from that of traditional catalyst support materials such as γ-alumina. γ-alumina is a high surface area, porous ceramic with a specific surface area, i.e. surface area per unit mass, of around 150 m^2/g and pore size as small as tens of Angstroms [10]. ZnO nanowires offer a morphology that is nearly the inverse of a traditional porous ceramic. Instead of having a solid material that is permeated with pores, the nanowires consist of solid, essentially two-dimensional objects, that protrude into the fluid in which they are immersed. In addition, the nanowires offered a material system with a relatively spatially uniform distribution of features such as porosity, nanowire spacing (analogous to pore size), nanowire length (analogous to support layer thickness), nanowire diameter, and low tortuosity. In ZnO nanowires, these properties can be precisely controlled so that the catalyst support material can be characterized well enough to provide inputs for the analytical model to fit the data. Procedures for growing ZnO nanowires on a wide variety of substrates are available in the literature [11, 12, 13, 14, 15]. Typical nanowire diameters produced by these methods range from 20 nm to 100 nm with lengths ranging from a few microns to 40 microns. In addition, ZnO has excellent thermal stability [16, 11], making it appropriate for the high temperature engine exhaust environment.

The concept of using substrates with nanostructures, including nanowires, as combustion catalyst supports has been studied previously with some success by Guo *et al.* [17]. In their novel investigation, Guo *et al.* synthesized SiC nanowires to serve as supports for Pd to be used in methane oxidation. They discovered that, due the presence of periodic stacking faults along the length of the nanowires, small

circumferential ditches, or grooves, can be etched in the nanowires. These grooves serve as a geometrical barrier to Pd agglomeration, thus preserving uniform Pd distribution within the nanowire support. The work presented here did not explore the use of crystallographic flaws in the nanowires as a means of inhibiting catalyst metal agglomeration, but this work did provide a useful means of experimentally validating an analytical model that was developed concurrently with the experimental work.

Nanowires will likely have different pore-region diffusion of reactants in the catalyst media because they can have greater effective porosity and average diffusion length scale than traditional γ-alumina. Numerous studies by Coppens's group have demonstrated the importance of optimizing pore morphology. These studies have used analytical and numerical models to explore the effects of fractal pore geometries, bimodal pore size distributions, continuous pore size distributions, and other pore morphology parameters on diffusion in porous substrate media [18, 19, 20, 21]. Coppens *et al.* sometimes refer to certain nanoscopic fractal structures as fjords. In light of the fact that Coppens in most likely Scandinavian, this is somewhat humorous, and probably intentional.

For catalysts, there are three critical rate controlling regimes: bulk diffusion, pore diffusion, and chemical kinetics. For a given catalyst with a constant volumetric throughput, temperature is what determines which regime is the dominant limiting mechanism for species conversion. A conceptual example of how the controlling regimes might operate for a typical automotive exhaust species is shown in Figure 1.2.

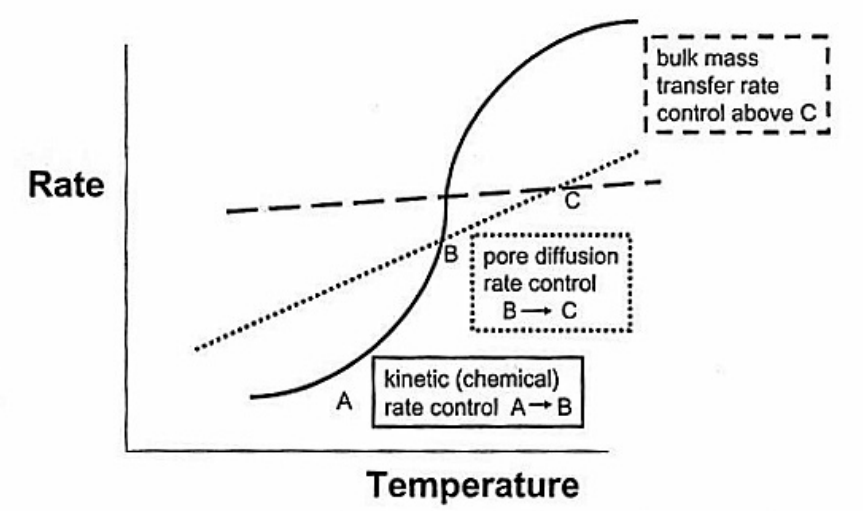


Figure 1.2: Conceptual plot of species conversion efficiency v. temperature. At low temperature (A to B), the limiting mechanism is chemical kinetics on the surface of the catalyst particles, at mid-range temperature (B to C), the limiting mechanism is pore diffusion of the reactant species in the catalyst support, and at high temperature (above C), the limiting mechanism is bulk diffusion of reactant species in the gas flow. This image was taken from Heck and Farrauto [10].

This plot is possibly misleading because it may give the impression that the rate limiting mechanism is the only mechanism that effects species conversion, but this is not the case. While it is true that these mechanisms are not additive, when the rate of one mechanism is on the order of that of another, both mechanisms should be considered important in determining the overall species conversion. The open pore geometry and possible high specific surface area of nanowires may offer performance benefits in mid-range temperatures where pore diffusion is the rate controlling mechanism. The level of control over the nanowire length, which is in the five to ten micron range, offers another difference from traditional γ-alumina support material, which has a typical wash-coat thickness of tens of microns ranging up to hundreds of microns in the corners of a square catalyst channel [10].

One complication associated with the use of zinc oxide is the possibility of

strong metal support interaction (SMSI) between the nanowires and catalyst metal affecting hydrocarbon oxidation and/or sintering performance [22]. SMSI can explain a range of effects including hindering the rate of catalyst metal agglomeration, increasing the rate of catalyst metal agglomeration, hindering the rate of species conversion, or increasing the rate of species conversion. The affect of SMSI on hydrocarbon oxidation was not directly addressed in this research because this type of SMSI effect was expected to be small since the fraction of catalyst sites on each Pt/Pd particle that were in contact with the nanowire surface was small. Also, this research was not intended to explore the industrial usefulness of ZnO nanowires as a catalyst support so agglomeration, i.e. sintering, performance was not studied.

While empirical models, semi-empirical models, numerical models, and commercial multi-physics packages have been used to predict catalyst behavior, the utility of two-dimensional analytical models that predict catalyst performance as a function of channel height and length and species concentration in both stream-wise and transverse dimensions has been sparsely addressed in the literature. Many previous studies have modeled catalyst performance using numerical and/or empirical models. A semi-empirical model for three-way catalysts with transient sub-models and empirical mapping for species conservation was presented by Laing *et al.* at Ford Motor Company [23]. Numerous other models have accounted for heat transfer, transient behavior, oxygen storage, and other effects. Koltsakis *et al.* developed a model that considers transient heat transfer and transient temperature-dependent kinetics. The species concentration was spatially lumped in the transverse direction so only stream-wise transport was considered [24]. Lambert *et al.* at Ford Motor Company have done modeling and experimental investigations of NO_x catalysts in compression ignition engines. Their goal was to develop a model that can be integrated in a larger vehicle systems model. The model, known as SIMTWC, predicts

catalyst performance as a function of volumetric throughput and temperature; the model accounts for heat transfer, transient species storage, and chemical kinetics using empirical submodels [25]. Heck and Farrauto have shown that a one-dimensional model that predicts stream-wise species concentration that is lumped in the transverse direction can provide relatively accurate results, but this model requires the Sherwood number, a dimensionless mass transfer coefficient that is equivalent to the Nusselt number, and other non-dimensional numbers that have been obtained from empirical studies [10].

There has been some work in catalyst optimization using combined analytical and/or numerical models with empirical maps of operating conditions by Katare *et al.* [26, 27]. These efforts are important for the design and manufacturing of industrial catalysts where rapid, accurate results are needed in order to meet tightening emissions standards and the rapid pace of production. This type of work would likely benefit from an accurate, fast, analytical species transport model because such a model would reduce the number of experiments needed to determine the optimal catalyst design.

Bhattacharya *et al.*, as part of the catalyst group in Chemical Engineering at University of Houston, were the first to address the need for a two-dimensional model that coupled species convection in the stream-wise and transverse directions in the channel of a catalyst with species diffusion and reaction in the wash-coat [28, 29]. They solved the partial differential equation (PDE) governing the physics of convection in the channel using separation of variables and a Fourier series expansion. They specifically state in their paper that this modeling technique has not previously been published in the open literature. This model also included the effects of various wash-coat shapes and channel shapes. Their model solved the PDE for discrete values

of chemical kinetics, and they could model continuous kinetics only for asymptotic conditions with extreme values of transverse Peclet number near zero or infinity.

Work done by Joshi *et al.* in the same group has demonstrated the potential usefulness of a low-dimensional model for real-time modeling and control of vehicle emissions systems [30, 31, 32]. This work presented a means of analytically determining Sherwood number based on duct geometry, flow rate, temperature, wash-coat geometry, wash-coat properties, and catalyst properties. One way Joshi *et al.* validated their low dimensional model was by comparing it to the PDE model developed earlier by Bhattacharya *et al.*, which was limited either to asymptotic cases in which the Peclet number was either very large or near zero or discrete values of kinetics. The Joshi model was validated for only a single operating condition using this method. As another means of validating their model, Joshi *et al.* used a COMSOL model, but this was used for only a single value of kinetics as finite element modeling is a cumbersome tool for exploring a large space of operating conditions and configurations. This work is valuable for several reasons: parabolic flow is considered; the 1D model can be used over a range of flow and temperature operating conditions; and the model provides a lot of insight into which mechanism, out of kinetics, pore diffusion, or bulk diffusion, is the most dominant in limiting species conversion. However, a significant limitation of this work is that the model is not capable of accounting for transverse effects in conditions where they are important, i.e. high Peclet number conditions, because it is only 1D. For high Peclet number conditions, entrance effects become important, and a two-dimensional model is needed for an accurate solution. In all of the work mentioned in this paragraph, the authors never presented an algorithm to systematically solve for the eigenvalues needed to solve the PDE governing the species concentration in the catalyst channels, and as of this writing, no such model exists in the open literature.

1.3 Intellectual Merit

The modeling portion of this work solves the species transport PDE in the catalyst channel and couples it to the reaction/diffusion ordinary differential equation (ODE) in the nanowire (porous media) region. The PDE is solved using separation of variables. The model development is similar to that of Bhattacharya *et al.* [28]; however the solution technique has been extended to allow for non-discrete, continuous values of Thiele modulus, i.e. a dimensionless ratio of reaction rate to diffusion rate within the substrate media, over a continuous range. This is the result of a systematic algorithm that can rapidly solve for the eigenvalues needed to solve the PDE so that this modeling technique can be as useful as, and potentially more accurate than, the models presented by Bhattacharya *et al.* and Joshi *et al.* A procedure to solve the PDE for continuous values of kinetics and Peclet number is a key contribution of this work. This will be discussed in Section 4.2.2.

The experiments were used to empirically validate the model. In all experiments and modeling, the hydrocarbon under consideration was propane, and assumed first order kinetics were assumed [33].

Chapter 2

Experimental Apparatus and Methods

The experimental portion of this work is composed of substrate preparation, characterization of substrates, and catalyst performance. Substrate preparation consists of nanowire growth and catalyst metal deposition. Characterization of substrates was done to determine the morphology of the nanowires and the uniformity of the catalyst metal deposition. Catalyst performance testing was carried out to evaluate the propane oxidation performance of the catalysts in an oxygen rich environment over a range of temperature and flow rate.

The usage of the word substrate in this text can vary depending on context. In the context of depositing catalyst metals on a substrate, substrate is usually referring to ZnO nanowires or γ-alumina on which the catalyst metal was deposited. In the context of nanowire growth, substrate refers to the Si wafer on which the nanowires were grown.

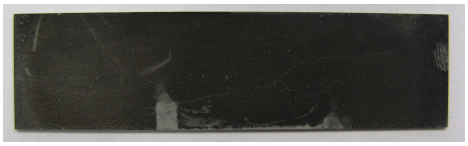
2.1 Substrate Preparation

Substrate preparation consisted of growing nanowires and then coating catalyst metals on substrates. The nanowire growth will be discussed first, followed by the catalyst metal coating.

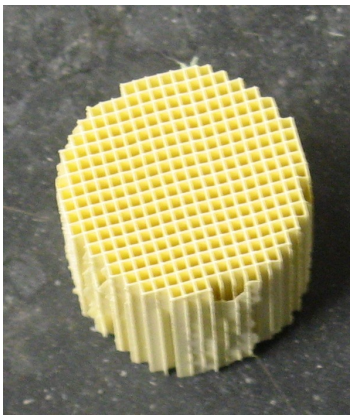
2.1.1 Nanowire Growth

The nanowire growth was initially carried out using a vapor transport procedure, but this proved to be ineffective for scalable, uniform, consistent nanowire growth and will not be discussed here. For the interested reader, please see the author's MS thesis [34]. For the purpose of achieving spatially uniform, repeatable, macroscopically scalable nanowire synthesis, aqueous solution growth proved effective, and this was used for all experiments in this dissertation.

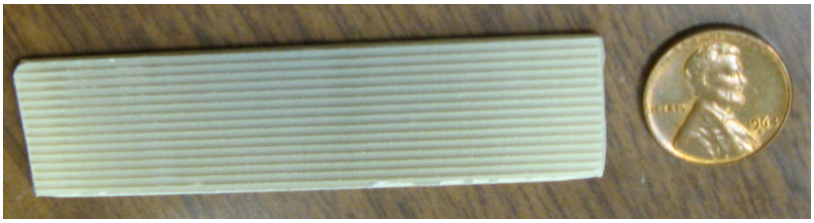
Two types of substrates were used for this work. First, in order to provide a reasonably simple geometry for growth, flat Si wafers were used. Later, Ford Motor Company donated numerous cordierite honeycomb monoliths, some of which were wash-coated by Ford with γ-alumina; these were used for control experiments. Some of the cordierite cores were machined down to wafer geometry to match the geometry of the Si wafers. A photo of a typical Si wafer, after growth, is shown in Figure 2.1a, a photo of a typical cordierite core is shown in Figure 2.1b, and a photo of a machined cordierite wafer is shown in Figure 2.1c.



(a) Si wafer after growth.



(b) Cordierite core.



(c) Cordierite wafer.

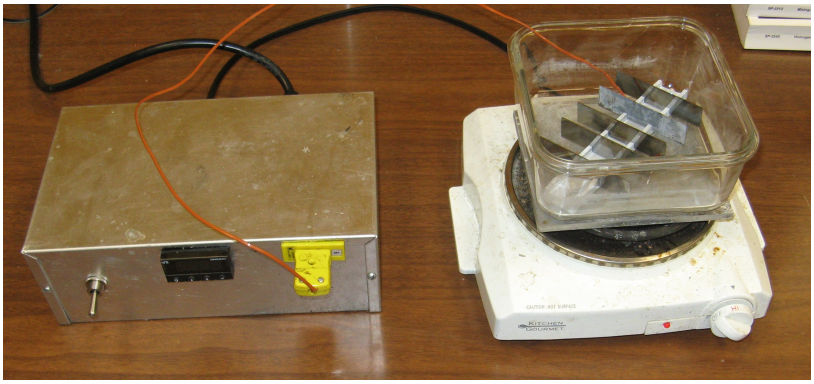
Figure 2.1: Photographs of (a) Si wafer, (b) cordierite core, and (c) machined cordierite wafer.

The Si wafers were diced from larger $\langle 100 \rangle$ wafers using a Disco DAD-321 dicing saw.

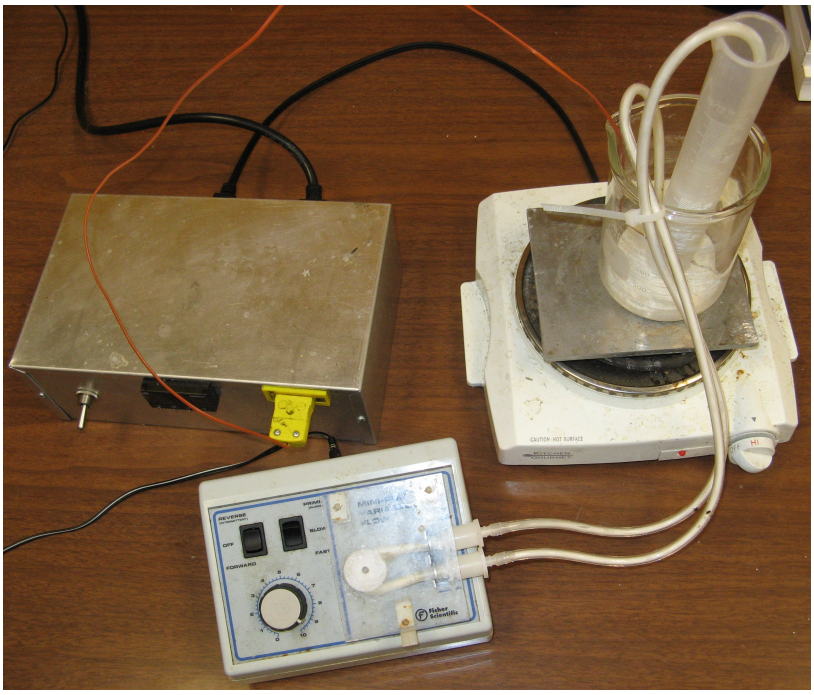
The apparatus used to grow the nanowires consisted a basic rig with some additional components that were used depending on which type of growth was desired. The rig for both types of growth is shown in Figure 2.2. For nanowire growth on the Si wafers, the rig consisted of a hot plate, a heat-spreading aluminum plate, a temperature controller with a K-type thermocouple, and the substrate-specific growth container. All of this is shown in Figure 2.2a.

The other configuration, consisting of the same basic components with the addition of a Fisher Scientific variable flow rate tube pump and a different growth container, was used for growing nanowires in the cordierite honeycomb cores and is

shown in Figure 2.2b. The pump was used to induce a gravity driven flow through the square channels of the cordierite honeycomb core. This ensured that convection, rather than merely pure diffusion, was delivering the ZnO precursor chemicals to the growth sites.



(a) Wafer growth rig.



(b) Honeycomb core growth rig.

Figure 2.2: Photograph of aqueous solution growth apparatus for (a) wafers and (b) honeycomb cores.

Zinc oxide nanowires were grown using an aqueous solution technique from several papers by Greene *et al.* [12, 13, 14]. The precursor chemicals used for the growth were zinc acetate for ZnO seed formation, zinc nitrate hexahydrate for nanowire growth, hexamethylenetetramine (HMTA) to serve as a pH buffer, and polyethylenimine (PEI) to hinder lateral growth of the nanowires. First growth sites for the nanowires had to be created. To do this, an ethanol solution with 25 mM of zinc acetate dihydrate was prepared. Next, the zinc acetate solution was dropped onto a 20 mm x 76.2 mm Si wafer and allowed to remain on the wafer for 10 seconds. The wafer was then rinsed with pure ethanol and blown dry with air. These steps were repeated four times for a total of five drop, rinse, dry cycles. After the zinc acetate washing process was complete, the wafer was placed into a furnace at 300 °C for 30 minutes. This created ZnO seeds that would subsequently serve as growth sites.

The growth process occurred in an aqueous solution of 500 mL deionized water that was prepared with 25 mM zinc nitrate hexahydrate, 25 mM HMTA, and 5 mM PEI. The solution was placed in a beaker on top of a hot plate, and the wafer was placed facing slightly downward at a $\sim 85^{\circ}$ angle from the bottom of the container. Placing the wafer facing slightly downward at an angle prevented bubbles from forming on the downward facing growth surface while also preventing precipitate from accumulating on the growth face. The hot plate was set to maintain the bath at 90 °C for three hours, and this was repeated two times with fresh solution to ensure long nanowires were grown. After the bathing process, the surface was rinsed using tap water to remove any precipitate from the nanowires.

For the cordierite cores, the seeding process was altered because drops could not be placed on the inner surfaces of the honeycomb structure. Instead, the cores were immersed in the seeding solution then immersed in a rinse solution, and this was

repeated four times as with the seeding method for the wafers. For the growth, the rig shown in Figure 2.2b was used to pump the growth solution through the channels of the honeycomb. All other aspects of the nanowire growth were the same as for the wafers.

2.1.2 Catalyst Metal Deposition

Sputter coating was the primary method used for coating the support materials with catalyst metals. A sputter coater directs an argon plasma generated by radio waves in high vacuum onto a sputter target, which in the case of this work, was a 80% Pt / 20% Pd. The plasma dislodges ~10 nm catalyst metal particles from the surface of the sputter target, which then fall onto the catalyst substrate, coating the substrate material through a process that is a mix of diffusion and line-of-sight ballistic transport. A Cressington 208HR sputter coater (located in the Institute for Cellular and Molecular Biology (ICMB) lab in the Molecular Biology Building (MBB)) was used for the coating in this work. A photograph of this sputter coater is shown in Figure 2.3.

Figure 2.3: Photograph of sputter coater [35].

All of the wafers, Si and cordierite, were coated using the sputter coater. The wafers were placed in the sputter coater, the user programmed in the desired sputter settings (current: 20 mA and density: 19.52 g/cm^3 for Pt/Pd) and deposition thickness, and then the process was automated. The automated process consisted of pumping a vacuum on the order of 10^{-6} torr in the sputter chamber, purging with Ar gas, then reestablishing the vacuum with a slow intentional Ar leak to provide atoms for ionization in the plasma. The deposition thickness was monitored and controlled using a quartz crystal deposition monitor. The quartz crystal deposition monitor is a disc that is coated during the sputter process, and as more material is deposited on the quartz surface, the frequency at which the quartz vibrates is reduced. The mass of material that is deposited on the surface can be determined based on the change in resonant frequency of the quartz, and the film thickness can be determined if the material density is known. This thickness is based on an assumed uniform film of sputtered material. In the context of sputter coating nanowires, the sputter thickness is the amount of material that would form a film of the desired thickness if it were uniformly deposited on a flat surface without any nano-scale texture. This method for coating the catalysts with Pt/Pd was chosen because it provided an easy way to determine the loading of catalyst metal.

In addition to sputter coating, which was used solely for the wafers, solution impregnate the cordierite cores with chloroplatinic acid (CPA) or tetraaimmineplatinum(ii) hydroxide (TPH) aqueous solutions was attempted. This method of coating was attempted exclusively for the cylindrical cordierite honeycomb cores. The wafers were immersed in an aqueous solution of either CPA or TPH. Citric acid or ammonia was used in an attempt to tune the pH of the solution to the point of zero charge (PZC) to allow effective adsorption of the precursor chemical on the substrate surface. The PZC is the pH of the solution at which the net charge density on the substrate

surface is zero. The pH was estimated based on the molarity of the pH-tuning solute in the aqueous solvent. The reported PZC values for TPH and CPA are substrate dependent and vary substantially in literature, and no values were found specifically for ZnO nanowire substrates [36]. The Pt precursor solute concentration was unknown because only a small portion of the solid Pt precursor chemical dissolved in the aqueous solution. This method was never successfully used to coat a substrate with Pt.

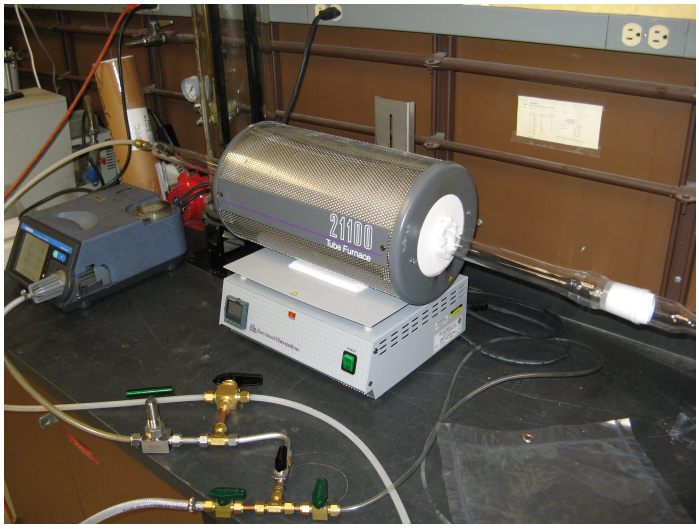
2.2 Material Characterization

The morphology of the substrates, support materials, and catalyst metal particles were characterized using a combination of optical microscopy, scanning electron microscopy (SEM), and transmission electron microscopy (TEM). For the SEM work, a Zeiss Supra Variable Pressure SEM (located in the Institute for Cellular and Molecular Biology (ICMB) lab in the Molecular Biology Building (MBB)) was used to observe support materials, nanowires, and large Pt/Pd particles. All of the TEM work was performed on a 120 kV FEI Tecnai (located in the Institute for Cellular and Molecular Biology (ICMB) lab in the Molecular Biology Building (MBB)) using a 400 mesh copper grid with a carbon film.

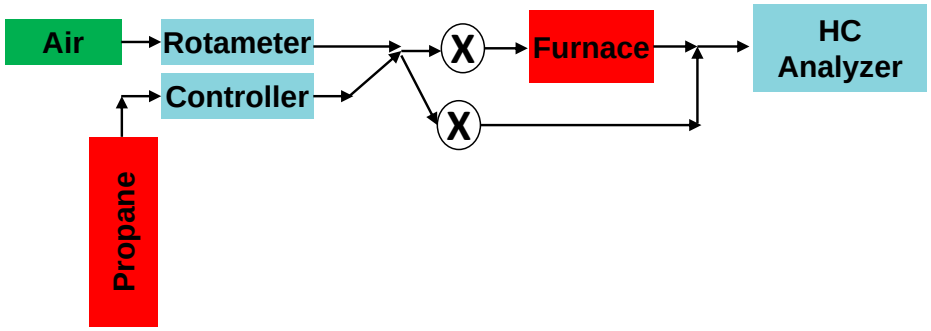
For the solution impregnation of Pt, no techniques to determine the dispersion and/or concentration of Pt on the support material of the honeycomb cores were available. As such, the solution impregnation procedure described in Section 2.1.2 was discarded as a viable method for affixing Pt to the support material for this work.

2.3 Catalyst Performance

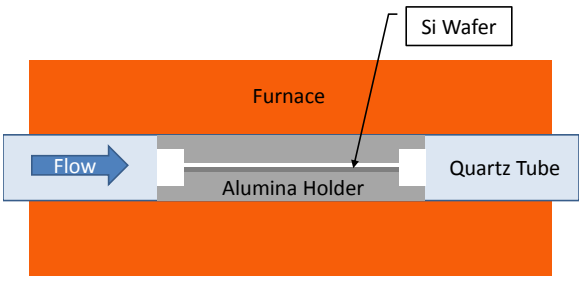
The catalyst performance testing was used to determine efficiency for reducing the concentration of emissions by sampling emission species concentration both upstream and downstream of the catalyst. The apparatus used for this testing are shown in Figures 2.4 and 2.5.



(a) Photo.

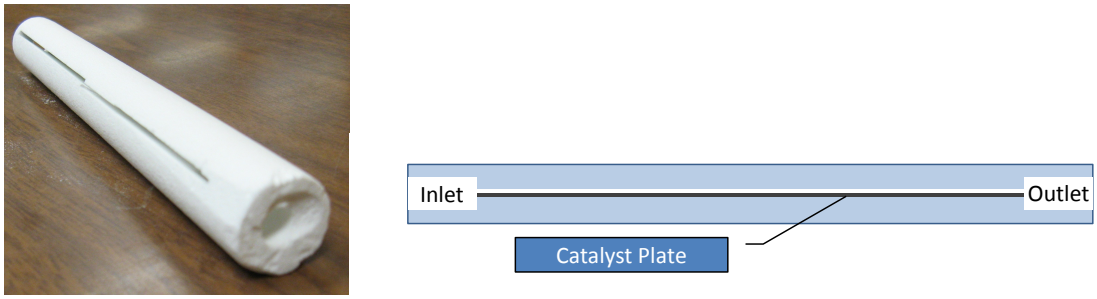


(b) Schematic.



(c) Furnace schematic.

Figure 2.4: (a) Photo and (b) schematic of catalyst test rig. (c) Detailed schematic of furnace.



(a) Photo. (b) Schematic.

Figure 2.5: Schematic of alumina sample holder.

The Si and cordierite wafers were held in place by a machined alumina cylinder (shown in Figure 2.5) with a slot that runs most of the length of the cylinder. Each end has a hole that allows exhaust gases to flow through the slot.

The alumina sample holder was placed inside a 22 mm internal diameter quartz tube in a tube furnace (shown in Figures 2.4a and 2.4c). 3/8 in. external diameter polyethylene tubes were used to flow simulated exhaust gases, consisting of a lean propane/air mixture, through valves (shown in Figure 2.4a) that could flow gases either through the catalyst or through a bypass system that would allow sampling both upstream and downstream of the catalyst. A Horiba Mexa-584L exhaust gas analyzer (shown in the far left of Figure 2.4a) that measured hydrocarbon, oxygen, carbon dioxide, and carbon monoxide concentrations was used for sampling the test gas. Propane was selected as the species of interest because we had convenient access to a sputter target containing primarily platinum, which is a good hydrocarbon oxidizing catalyst, and the exhaust gas analyzer had the highest sensitivity to hydrocarbons, particularly propane, which is the hydrocarbon used to calibrate the Mexa-585L.

The test gas in all cases consisted of a mixture of propane and standard air. The flow rate of propane was controlled by a linear electronic flow controller (make:

Aera, model: FC-7800XCU, range of flow rates: 0 to 30 sccm, input/output: 0 to 5 V), and the flow rate of the air was controlled by a calibrated rotameter (make: Expotech, model: G-03294-18). A plot of the rotameter calibration data and a linear curve fit is shown in Figure 2.6.

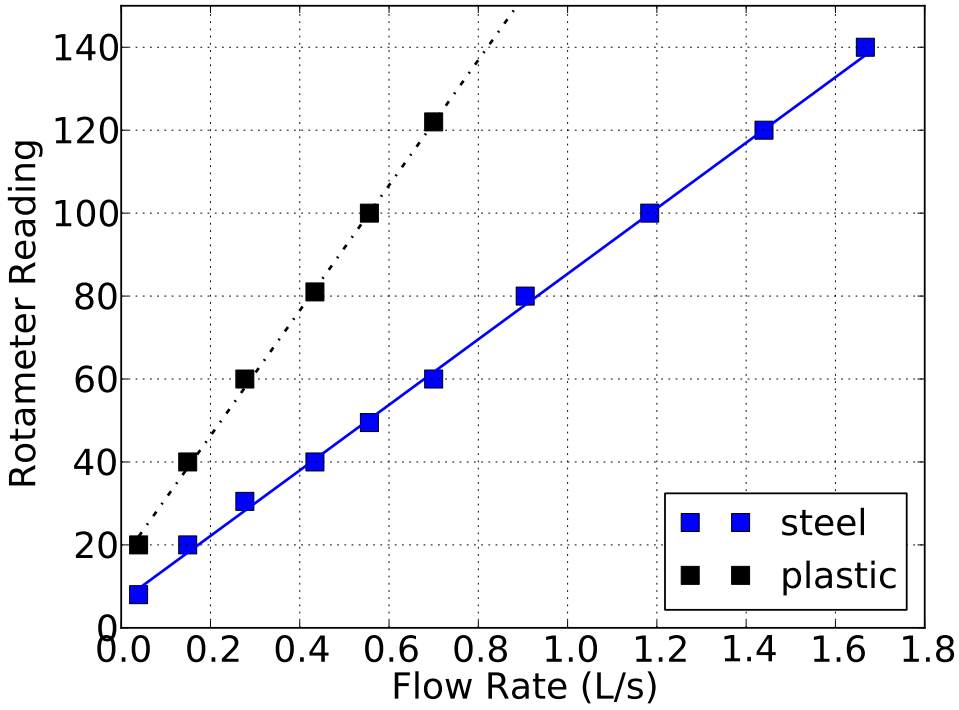


Figure 2.6: Plot of rotameter calibration data with linear curve fit. The label steel indicates the reading of heavier steel ball, and the label plastic indicates the reading of the lighter plastic ball.

Conversion efficiency was determined from the measured hydrocarbon concentrations upstream and downstream of the furnace. The hydrocarbon analyzer manufacturer's stated measurement accuracy was +/-5%, but the reproducibility of the measurements showed a precision uncertainty of less than 1%. The air flow was controlled using a calibrated rotameter with an uncertainty of +/-2% at 500 sccm which was the estimated uncertainty of the total flow rate since the contribution of

the propane flow rate uncertainty was negligible. The calculated uncertainty in the conversion efficiency was less than +/-2%. The temperature variation through the furnace was measured using thermocouples; this variation was less than 10 K. The inlet gas had a fuel-air equivalence ratio of approximately 0.1.

Chapter 3

Experimental Results

3.1 Substrate Characterization

SEM was used to determine the nanowire areal density, length, diameter, and an estimate of the porosity of the nanowire material; an SEM image of nanowires grown on a silicon wafer is shown in Figure 3.1.

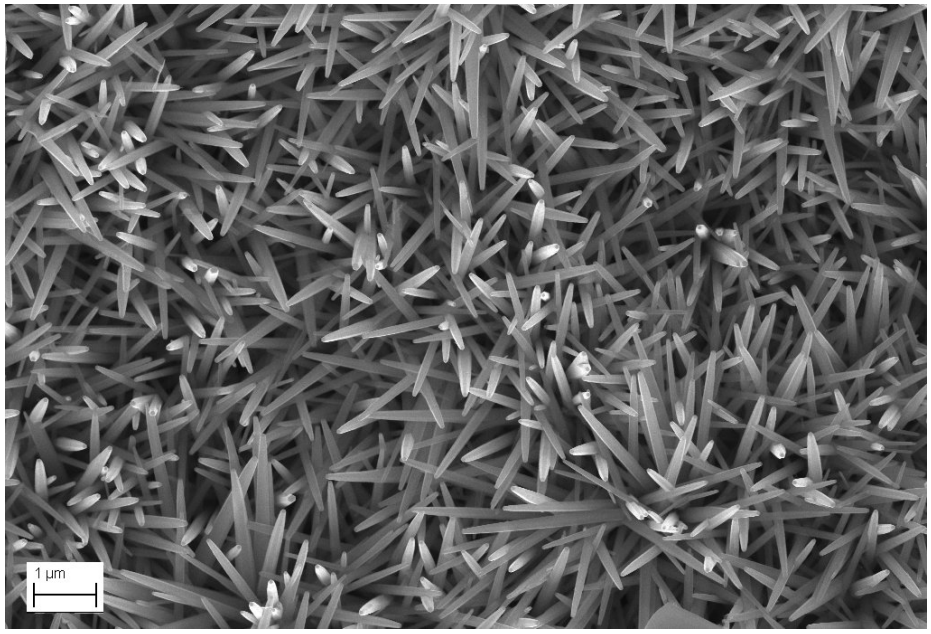


Figure 3.1: SEM image of nanowires grown on Si wafer.

These nanowires were typically 2 μm in length with an average diameter of approximately 100 nm. This results in a calculated specific surface area of 7 $\frac{\mathrm{m}^2}{\mathrm{g}}$. The nanowire coverage was quite dense over the entire substrate, with approximately 10 nanowires per 1 $\mu\mathrm{m}^2$.

TEM was used to qualitatively determine nanowire size, Pt/Pd particle size, and Pt/Pd particle distribution. A TEM image of a nanowire sputter coated with 1 nm Pt/Pd is shown in Figure 3.2.

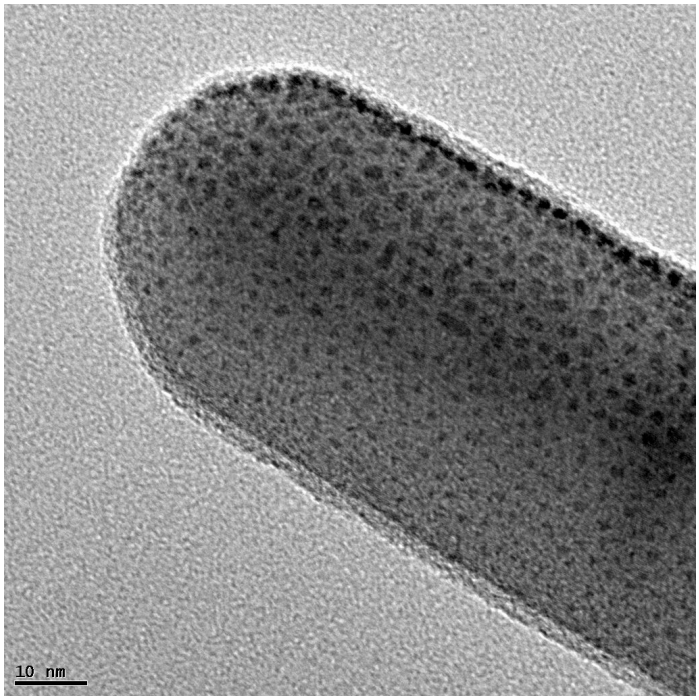


Figure 3.2: TEM image of nanowire after being sputter coated with 1 nm Pt/Pd.

In Figure 3.2, the average particle size is about 3 nm, and the average particle spacing is also about 3 nm. Sputter coating is a process that utilizes ballistic and diffusional transport of Pt/Pd particles from the sputter target (the source material) to the sample on which deposition is occurring. The result of the ballistic effects is that nanowire surfaces facing the sputter target are more densely coated than surfaces facing away from the sputter target, causing a shadow effect. This shadow effect can be seen on the nanowires in Figure 3.2.

3.2 Performance

3.2.1 Effect of Pt/Pd loading

The hydrocarbon analyzer was used to determine hydrocarbon conversion efficiency by measuring the percentage decrease in hydrocarbon concentration downstream versus upstream of the catalyst. Conversion efficiency was found to be a function of the sputter thickness of the Pt/Pd layer. As an example, a plot of hydrocarbon conversion efficiency versus Pt/Pd sputter thickness at 450 °C and 500 sccm of fuel/air flow rate for Si wafers is shown in Figure 3.3.

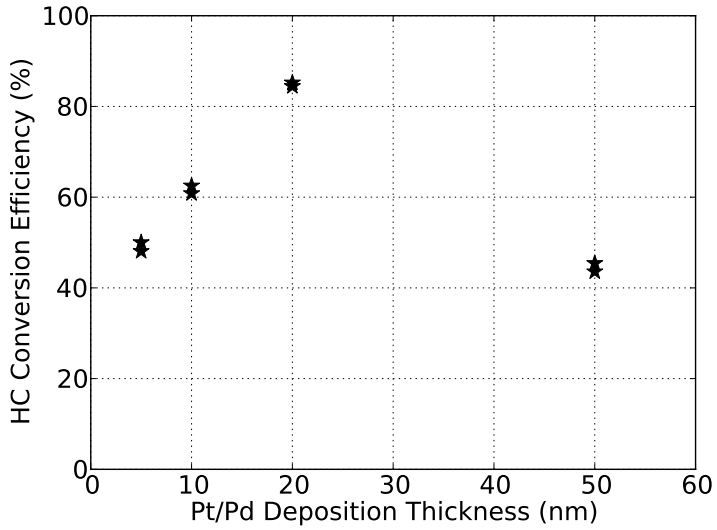
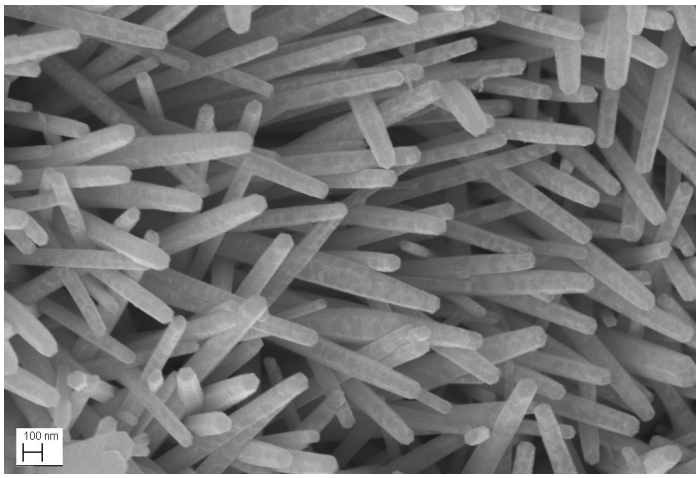


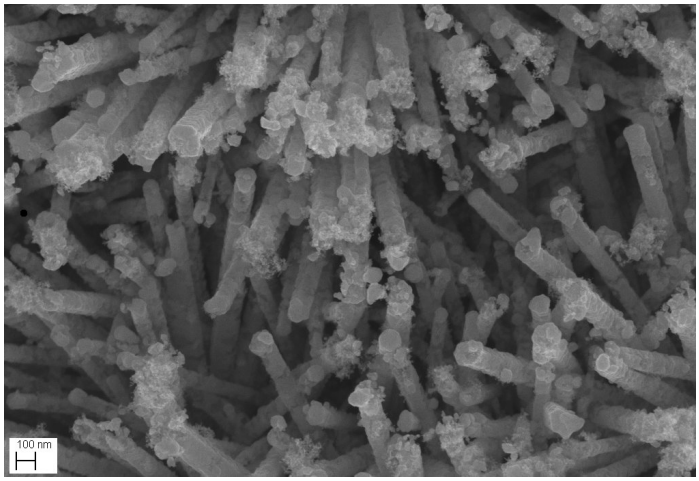
Figure 3.3: Plot of conversion efficiency v. Pt/Pd loading on nanowires on Si wafers at 450 °C and 500 sccm.

Figure 3.3 shows that increasing Pt/Pd loading initially increases conversion efficiency until 20 nm of Pt/Pd loading; then the conversion efficiency decreases with further Pt/Pd deposition. A likely cause of this precipitous decrease in conversion efficiency is agglomeration of Pt/Pd particles during the deposition process, resulting in larger particles, which reduces the total amount of exposed area of the particles.

The larger particles also might have reduced surface kinetic rates because larger particles tend to expose crystallographic planes that have less favorable conditions for catalytic activity [37]. SEM images showing the increase in particle agglomeration associated with increasing Pt/Pd sputter amounts are shown in Figures 3.4a and 3.4b.



(a) 20 nm Pt/Pd.



(b) 50 nm Pt/Pd.

Figure 3.4: SEM images showing how Pt/Pd agglomeration is affected by sputter thickness. Scale bars are 100 nm.

As observed in Figures 3.4a and 3.4b, for the case of higher metal loading, the Pt/Pd particles begin to agglomerate and lose available surface area.

3.2.2 Effect of Flow Rate

The effect of temperature and flow rate on conversion efficiency is shown in Figure 3.5. Shown are three flow rates for Si wafers with nanowires sputter coated with 10 nm Pt/Pd.

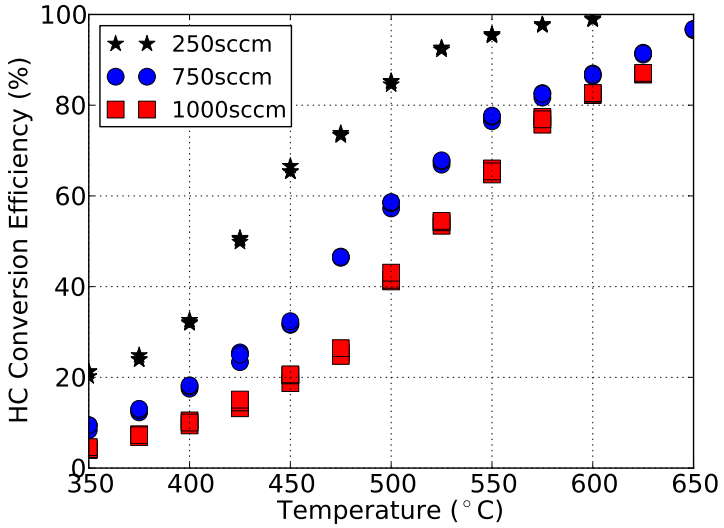


Figure 3.5: Plot of conversion efficiency v. temperature for three flow rates for Si wafers coated with nanowires with 10 nm Pt/Pd sputter deposition thickness.

In Figure 3.5, flow rates correspond to space velocities of 1640 hr^{-1}, 3281 hr^{-1}, and 4921 hr^{-1}. With increasing flow rate, the conversion efficiency is decreased because the residence time in the reactor is decreased. It is likely that the mass conversion rate is unchanged because kinetics and transverse diffusion are unaffected by flow rate.

3.2.3 Effect of Substrate Material

Figure 3.6 shows a plot of conversion efficiency versus temperature for the three different substrate materials: bare cordierite wafers, cordierite wafers coated

with γ-alumina, and nanowire-coated cordierite. All substrates have the same 10 nm Pt/Pd coating amount.

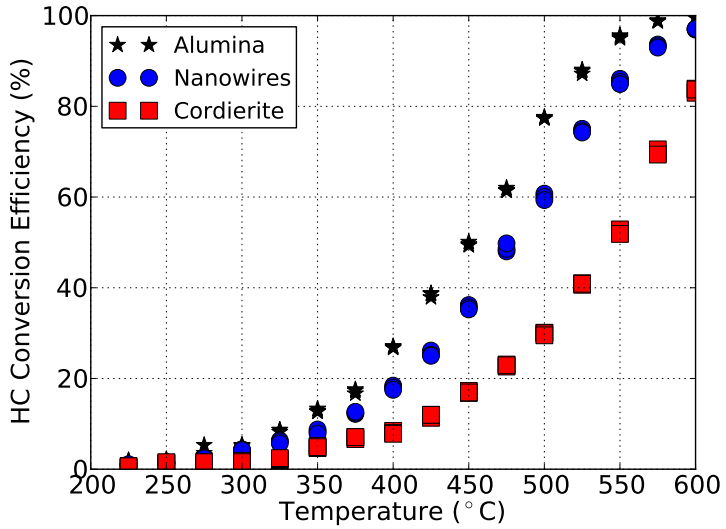


Figure 3.6: Plot of conversion efficiency v. temperature at 500 sccm for 10 nm Pt/Pd sputtered onto three substrate types: bare cordierite, γ-alumina-coated cordierite, and nanowire-coated cordierite.

In Figure 3.6, it can be seen that the hydrocarbon conversion efficiency of nanowires is much closer to that of γ-alumina than that of bare cordierite. The relatively better performance of the γ-alumina and nanowire surfaces over the bare cordierite is likely due to the higher surface area available for catalytic particle attachment in the absence of interaction and agglomeration. The nanowires were on the order of 100 nm in diameter; this corresponds to a specific surface area of 7 $\frac{\mathrm{m}^2}{\mathrm{g}}$, which is quite small compared to the industry standard of around 150 $\frac{\mathrm{m}^2}{\mathrm{g}}$ for γ-alumina [10].

One shortcoming of sputter coating as a sample preparation technique is that the γ-alumina, which is up to several hundred microns thick, likely had Pt/Pd coating in only the first few microns of depth because of the partially ballistic nature of sputter

coating. This may have resulted in the similar performance of nanowires and alumina as catalyst support materials. On the other hand, if the Pt/Pd had been uniformly distributed throughout the depth of the γ-alumina wash-coat, the rate of diffusion would have become increasingly limited with increasing depth into the wash-coat. This diffusion limitation might have been a bottle neck for the overall performance as well. Because of the open pore geometry of the nanowires, it is likely that the Pt/Pd particles were utilized nearly uniformly throughout the depth of the nanowire region for catalytic reactions. To glean more insight into how substrate morphology can affect hydrocarbon species conversion, an analytical model that couples convective species transport in the channel with diffusive species transport and chemical reaction in the porous media was developed.

Chapter 4

Model Development

This analytical model solves the PDE governing diffusive and advective transport in the channels of a catalyst couple with the ODE governing diffusive transport and chemical reaction rate in the porous media of the walls of a catalyst. This model introduces an algorithm to systematically solve for the eigenvalues needed to solve the PDE so that results can be obtained for continuous, rather than discrete, values of chemical kinetics. This also enables the model to solve the PDE for arbitrary values of Peclet number. This model accounts for entrance effects in the species concentration profile and is thus potentially more accurate than a 1D model for high Peclet number conditions in which entrance effects are important. The most obvious limitation of the model is that it assumes a uniform flow profile, but as comparison with experimental results will show, this assumption does not noticeably affect the accuracy of the model.

4.1 Approach

The species concentration within the catalyst channel and porous media of a catalyst consisting of two parallel plates was modeled to understand the underlying transport physics governing the performance of the nanowire catalysts. A schematic of the system that was modeled is shown in Figure 4.1.

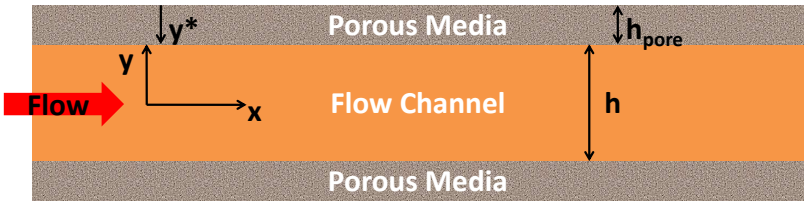


Figure 4.1: Schematic of catalyst model with coordinate system defined. Not to scale.

The purpose of this modeling effort was to develop a means of comparing different catalyst designs and to create a design tool for optimizing catalyst performance as a function of flow rate, temperature, channel geometry, wash-coat thickness, substrate morphology, catalyst metal loading, and catalyst metal activity. A benefit of this would be the ability to compare the performance of different real catalysts without relying on space velocity. The parallel plate model was used to predict the results for a flat catalyst plate in internal flow, as used in the experiments.

4.1.1 Governing Equations

A simplified form of the species conservation equation was applied to determine species concentration. Several assumptions allowed for reducing the complexity of the model. First, the reactant species were assumed to be sufficiently dilute that the energy released by the reactions did not affect the temperature of the flow. This also implies that the flow was isothermal, and therefore, heat transfer could be neglected. This assumption was validated by thermocouple measurements taken both upstream and downstream of the reactor during hydrocarbon conversion experiments, as mentioned earlier. Second, the hydrocarbon reactant was assumed to be sufficiently dilute that no depletion of oxygen occurred. These assumptions were consistent with the fuel lean conditions used in all of the experiments. The flow profile was assumed to be uniform across the channel. In reality, the laminar flow would develop to a parabolic

velocity profile, but accounting for this would not have significantly increased the predictive capability of the model, while adding to the mathematical complexity of the solution. Further, negligible bulk flow was assumed to occur in the porous media so transport in the porous media was only by molecular diffusion and only in the transverse direction [38]. Reactions were assumed to occur only on the catalyst metal surfaces in the porous media, and not in the gas phase. Because propane was the only species considered in the experimental results, the model also considered only propane. A first order kinetics reaction was assumed to be sufficient for modeling the propane reactions. The catalytic particles were assumed to be uniformly distributed throughout the porous media, and all catalyst particles were assumed to be the same size. Bulk, or channel, diffusion was assumed to occur in the transverse direction only, not the axial direction. Mass diffusivities were assumed to be spatially constant and constant with species concentration. With these assumptions, the governing equation for species (propane) concentration within the flow channel can be expressed as

$$\rho u \frac{\partial Y\left(x,y\right)}{\partial x} = \rho \mathscr{D} \frac{\partial^2 Y\left(x,y\right)}{\partial y^2} \tag{4.1.1}$$

In Equation 4.1.1, ρ is the gas density, u is the stream-wise bulk velocity, x is the stream-wise coordinate, $\mathscr{D}$ is the bulk diffusivity of the species of interest into air, Y is the mass fraction of the species of interest, and y is the perpendicular coordinate. The mass fraction, $Y\left(x,y\right)$, was resolved in both the stream-wise (x) and transverse (y) directions.

By defining a few scaling variables, Equation 4.1.1 can be simplified to a non-dimensional form. Scaling variables are as follows:

$$\tilde{y} = 2y/h \tag{4.1.2}$$

$$\tilde{x} = x/h \tag{4.1.3}$$

$$\tilde{Y} = Y/Y_0 \tag{4.1.4}$$

where $\tilde{y}$ is the dimensionless form of perpendicular coordinate y, h is the height of the flow channel $[m]$, $\tilde{x}$ is the dimensionless form of the stream-wise coordinate x, $\tilde{Y}$ is the scaled form of the species mass fraction Y, and Y_0 is the inlet mass fraction of the reactant species. The result of applying these scaling variables is a simplified equation for species concentration in the channel,

$$\frac{Pe_h}{4}\frac{\partial \tilde{Y}}{\partial \tilde{x}} = \frac{\partial^2 \tilde{Y}}{\partial \tilde{y}^2} \tag{4.1.5}$$

where $Pe_h = \frac{uh}{\mathscr{D}}$ is the transverse Peclet number with respect to duct height.

The equation for species concentration in the porous media is

$$\mathscr{D}_{\text{pore}}\frac{d^2Y}{dy*^2} = k_{\text{pore}}Y \tag{4.1.6}$$

where $\mathscr{D}_{\text{pore}}$ is the effective diffusion coefficient in the porous media, $y*$ is a coordinate system defined as zero at the edge of the porous media and increasing in the direction of the center-line of the flow channel, and $k_{\text{pore}}\left[\frac{1}{s}\right]$ is the effective volumetric reaction rate coefficient in the porous media. Equation 4.1.6 is a variation of the equation for mass balance in porous media given by [39]. Equation 4.1.6 can be non-dimensionalized by defining the following scaling variable:

$$\tilde{y}* = y*/h_{\text{pore}} \tag{4.1.7}$$

where $h_{\text{pore}}\,[\text{m}]$ is the height of the porous media.

The species concentration scaling variable is the same for the porous media and the channel. The non-dimensional form of Equation 4.1.6 is

$$\frac{d^2\tilde{Y}}{d\tilde{y}^2} = \phi\tilde{Y} \tag{4.1.8}$$

where $\phi \equiv \frac{k_{\text{pore}}h_{\text{pore}}^2}{\mathscr{D}_{\text{pore}}}$ is the Thiele modulus within the porous media [39].

4.2 Solution

4.2.1 Separation of Variables

Using separation of variables on the PDE (Equation 4.1.5) yields

$$\tilde{Y} = \sum_{n=0}^{\infty} \exp\left(-4\frac{\lambda_n^2}{Pe_h}\tilde{x}\right)\left(A_n \cos\left(\lambda_n \tilde{y}\right) + B_n \sin\left(\lambda_n \tilde{y}\right)\right) \tag{4.2.1}$$

where λ_n is the nth eigenvalue and A_n and B_n are the nth Fourier coefficients. The symmetry condition along the center-line of the channel, given by

$$\left.\frac{d\tilde{Y}}{d\tilde{y}}\right|_{\tilde{y}=0} = 0 \tag{4.2.2}$$

eliminates the sine term in Equation 4.2.1 which gives the final solution as

$$\tilde{Y} = \sum_{n=0}^{\infty} A_n \exp\left(-4\frac{\lambda_n^2}{Pe_h}\tilde{x}\right)\cos\left(\lambda_n \tilde{y}\right) \tag{4.2.3}$$

Conversion efficiency can be determined by subtracting the average value of species concentration at the outlet from the average value of the species concentration at the inlet. The expression for this is

$$\eta = \sum_{n=0}^{\infty} \frac{A_n}{\lambda_n} \sin\left(\lambda_n\right)\left[1 - \exp\left(-4\frac{\lambda_n^2}{Pe_h}\tilde{x}\right)\right] \tag{4.2.4}$$

where η is hydrocarbon conversion efficiency.

4.2.2 Applying Boundary Conditions

To determine the eigenvalues, λ_n, and the Fourier coefficients, A_n, the model for the channel must be coupled to the model for the porous media. To do this, two interface boundary conditions are needed. First, the species concentration must be the same for both regions at the interface,

$$\left.\tilde{Y}\right|_{\tilde{y}*=1} = \left.\tilde{Y}\right|_{\tilde{y}=1} = \tilde{Y}_{\text{int}} \tag{4.2.5}$$

where $\tilde{Y}_{\text{int}}$ is the channel species concentration at the interface between the channel and the porous media and is not yet known. For the next interface boundary condition, the species flux at the interface between the porous media and the channel must be equal,

$$\mathscr{D}\frac{dY}{dy}\bigg|_{y=h/2} = -\mathscr{D}_{\text{pore}}\,\frac{dY}{dy*}\bigg|_{y=h_{\text{pore}}} \tag{4.2.6}$$

or in non-dimensional form,

$$\frac{d\tilde{Y}}{d\tilde{y}}\bigg|_{\tilde{y}=1} = -\frac{h}{2h_{\text{pore}}}\frac{\mathscr{D}_{\text{pore}}}{\mathscr{D}}\,\frac{d\tilde{Y}}{d\tilde{y}*}\bigg|_{\tilde{y}*=1} \tag{4.2.7}$$

To solve the PDE (Equation 4.1.5) by applying Equation 4.2.7, the solution to the ODE (Equation 4.1.8) must be used. The solution is

$$\tilde{Y} = C_1\cosh\left(\sqrt{\phi}\tilde{y}*\right) + C_2\sinh\left(\sqrt{\phi}\tilde{y}*\right) \tag{4.2.8}$$

where C_1 and C_2 are integration constants that are not yet known. The outer edge of the porous media has an impermeable wall boundary condition,

$$\frac{d\tilde{Y}}{d\tilde{y}*}\bigg|_{\tilde{y}*=0} = 0 \tag{4.2.9}$$

Applying this boundary condition and the interface species boundary condition given by Equation 4.2.5, gives the solution to the ODE (Equation 4.1.8) as

$$\tilde{Y} = \frac{\cosh\left(\sqrt{\phi}\tilde{y}*\right)}{\cosh\sqrt{\phi}}\tilde{Y}_{\text{int}} \tag{4.2.10}$$

The derivative of Equation 4.2.10,

$$\frac{d\tilde{Y}}{d\tilde{y}*}\bigg|_{\tilde{y}*=1} = \sqrt{\phi}\tanh\left(\sqrt{\phi}\right)\tilde{Y}_{\text{int}} \tag{4.2.11}$$

provides a means of simplifying the interface species flux boundary condition (Equation 4.2.7) so that the interface can be treated as if it is a flat surface for the purpose

of providing a boundary condition for the PDE for the channel. Using Equation 4.2.11 in the interface species flux boundary condition (Equation 4.2.7) yields

$$\left.\frac{d\tilde{Y}}{d\tilde{y}}\right|_{\tilde{y}=1} = -\frac{h}{2h_{\text{pore}}}\frac{\mathscr{D}_{\text{pore}}}{\mathscr{D}}\sqrt{\phi}\tanh\left(\sqrt{\phi}\right)\tilde{Y}_{\text{int}} \tag{4.2.12}$$

Here, it becomes convenient to define a new non-dimensional parameter,

$$Da \equiv \frac{h}{2h_{\text{pore}}}\frac{\mathscr{D}_{\text{pore}}}{\mathscr{D}}\sqrt{\phi}\tanh\left(\sqrt{\phi}\right) \tag{4.2.13}$$

where Da is the surface Dahmköhler number. The interface species flux boundary condition can now be expressed as

$$\left.\frac{d\tilde{Y}}{d\tilde{y}}\right|_{\tilde{y}=1} = -Da\tilde{Y}_{\text{int}} \tag{4.2.14}$$

Applying the species flux interface boundary condition (Equation 4.2.7) to the solution of the PDE (Equation 4.2.3) yields

$$\sum_{n=0}^{\infty}\lambda_n A_n \exp\left(-4\frac{\lambda_n^2}{Pe_h}\tilde{x}\right)\sin\left(\lambda_n\right) = Da\sum_{n=0}^{\infty}A_n\exp\left(-4\frac{\lambda_n^2}{Pe_h}\tilde{x}\right)\cos\left(\lambda_n\right) \tag{4.2.15}$$

This can be rearranged and simplified to

$$\frac{\lambda_n}{Da} = \cot\left(\lambda_n\right) \tag{4.2.16}$$

which provides the solution for the eigenvalues. Equation 4.2.16 has been graphically solved in the literature [40].

The boundary condition for species concentration in the channel entrance is

$$\left.\tilde{Y}\right|_{\tilde{x}=0} = 1 \tag{4.2.17}$$

[40] provides an expression for the Fourier coefficients based on this boundary condition,

$$A_n = \frac{2\sin\lambda_n}{\lambda_n + \sin\lambda_n\cos\lambda_n} \tag{4.2.18}$$

To rapidly solve the model for an arbitrary value of Da, eigenvalues were tabulated as a function of Da, and a spline interpolation function was used to estimate the eigenvalues, λ_n. The result from this spline interpolation was then used to provide a starting estimate for a numerical solver that determined the exact eigenvalues with a high level of accuracy. The numerical solver could not be used directly because the eigenvalues have asymptotic behavior, and thus, the numerical solver needed accurate starting estimates for the eigenvalues. This technique enabled the model to run over a continuous range of varied Da without the need for user interaction which allowed the use of non-discrete, continuous values for chemical kinetics and continuous values of Pe_h. A plot of the numerical solution for the first four eigenvalues over a continuous range of Da with the graphically-determined discrete values is shown in Figure 4.2.

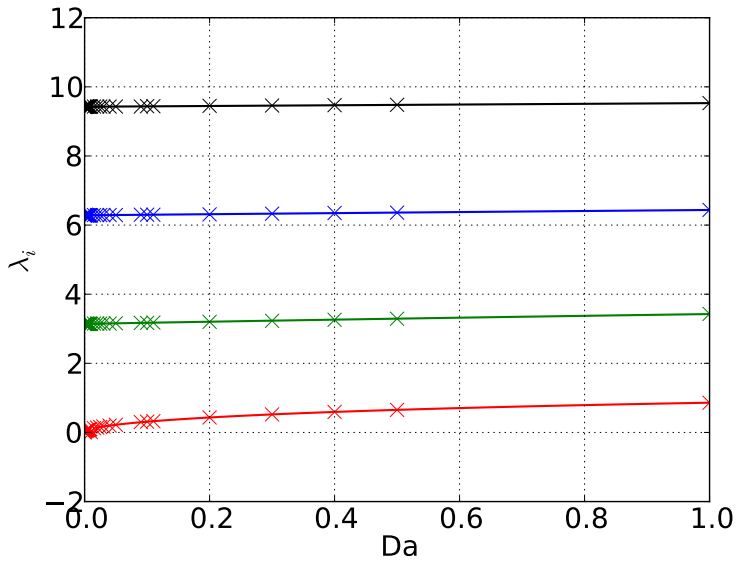


Figure 4.2: Plot of the numerical solution for the first four eigenvalues over a continuous range of Da with the graphically-determined discrete values.

4.2.3 Closure for Transport and Kinetics Parameters

The solution for the coupled differential equations is known, but mass diffusivity and chemical kinetics parameters remain unknown. The bulk mass diffusivity and mean free path of the reactant species in the gas can be estimated using theory presented in Transport Phenomena [41]; this is discussed in detail in Appendix A. The diffusion coefficient in the porous media can be estimated using a geometric average for diffusion in porous media presented by Wang *et al.* [18],

$$\mathscr{D}_{\text{pore}} = \frac{2\epsilon}{\tau} \frac{\mathscr{D}\mathscr{D}_{\text{Kn}}}{\mathscr{D} + \mathscr{D}_{\text{Kn}}} \tag{4.2.19}$$

where ϵ is the porosity of the substrate material and τ is the tortuosity of the porous media, which is a measure of how much the pores in a material deviate from being perfectly straight. A tortuosity of zero means the pores are perfectly straight, and a high value of tortuosity means the pores are highly curved and/or kinked. $\mathscr{D}_{\text{Kn}}$ is the Knudsen diffusivity, defined as

$$\mathscr{D}_{\text{Kn}} \equiv \begin{cases} \frac{\mathscr{D}}{\text{Kn}}, Kn > 1 \\ \mathscr{D}, Kn \leq 1 \end{cases} \tag{4.2.20}$$

which is the bulk diffusivity divided by the Knudsen number when the Knudsen number is greater than one. Note that the two in the numerator of Equation 4.2.19 is not present in Wang *et al.*. The two is necessary because if $\mathscr{D}$ and $\mathscr{D}_{\text{Kn}}$ are equal, then $\mathscr{D}_{\text{pore}}$ should also be the same value, but this is not the case if the factor of two is not present in the numerator. The Knudsen number is defined as $Kn \equiv \frac{\lambda}{L}$, where λ is the mean free path of the gas species and L is the characteristic length scale associated with the geometry of the catalyst.The tortuosity was unknown for the ZnO nanowire substrates so it was modeled as $\tau \approx 1/\epsilon$ as suggested by Wang *et al.* [18]. Porosity was estimated by counting nanowires in an SEM image and dividing the total cross-section area of the nanowires by the total area of the image. The porosity

was 97% based on the SEM image in Figure 3.1. The inter-nanowire spacing was used as the length scale for calculating the Knudsen number, and this resulted in a Knudsen number that was always less than unity. Therefore, the Knudsen diffusivity was assumed to be equal to bulk diffusivity in Equation 4.2.19.

The chemical kinetics parameters remain unknown, and the model accounted for this by assuming a first order Arrhenius kinetics rate constant of the form

$$k_{\text{pore}} = A \exp\left(\frac{-T_a}{T}\right) \tag{4.2.21}$$

where A is a pre-exponential fit parameter and T_a is the activation temperature for the reaction. The pre-exponential fit parameter and activation temperature were determined by fitting the data to conversion efficiency over a range of temperatures for a fixed flow rate.

To further increase the depth of the model, the surface kinetics can be estimated if the average catalyst particle size is known,

$$k_{\text{pore}} = k_{\text{part}} n_p A_p \exp\left(\frac{-T_a}{T}\right) \tag{4.2.22}$$

where $k_{\text{part}}\left[\frac{1}{s}\right]$ is the particle reaction rate coefficient, $n_p\left[\frac{\#}{m^3}\right]$ is the number of catalyst particles per unit volume, and $A_p\,[m^2]$ is the exposed surface area per catalyst particle. In Equation 4.2.22, a catalyst particle is a typical Pt/Pd nano-particle as shown on the nanowire in Figure 3.2. The dark dots are images of the particles. The Pt/Pd deposition thickness was used as a surrogate variable for catalyst metal particle loading. Equation 4.2.22 is presented as means of showing how average kinetics at the individual particle level might be determined if particle density and surface area are known. It also justifies the scaling used in Section 5.2.2 because the pre-exponential parameter in Equation 4.2.21 can be scaled by the particle number density, n_p.

The model is now fully closed and can be used to calculate conversion efficiency or species concentration in two dimensions in the porous media and channels of a catalyst as a function of flow rate, temperature, channel height, channel length, wash-coat thickness or nanowire length, catalyst particle size, and catalyst particle surface kinetics.

4.3 Meaning of Dimensionless Parameters

Before proceeding, it is pedagogically important to discuss the significance of the various non-dimensional parameters associated with this model. The non-dimensional parameters are highly useful because they provide a means of discerning which physical mechanisms (chemical kinetics, pore diffusion, and/or bulk diffusion) are most important in limiting the performance of a catalyst. The relevant non-dimensional parameters are the transverse Peclet number, Pe_h, the Dahmköhler number, Da, and the Thiele modulus, ϕ. Pe_h is the ratio of stream-wise advection relative to transverse (bulk) diffusion, Da is the ratio of effective surface chemical kinetics at the wall relative to transverse (bulk) diffusion in the flow, and ϕ is the ratio of effective volumetric chemical kinetics in the porous media (nanowires or γ-alumina wash-coat) relative to pore diffusion. Based on these definitions, Table 4.1 shows some useful observations that can be made from these non-dimensional parameters. k_{surf} is the effective wall reaction rate, or the numerator of Da, and these are also shown graphically in Figure 4.3.

Table 4.1: Non-dimensional parameters and corresponding rate limiting mechanisms. For each condition of each parameter, the rate coefficient that is not limiting is listed, e.g. $!\mathscr{D}_{\text{pore}}$ means that $\mathscr{D}_{\text{pore}}$ is not (!) the limiting mechanism for the corresponding condition. This enables elimination of the mechanisms that are not rate limiting so that only the remaining mechanism(s) is/are limiting.

Pe_h $\left[\frac{\text{advection rate}}{\text{bulk diffusion rate}}\right]$	Da $\left[\frac{\text{wall reaction rate}}{\text{bulk diffusion rate}}\right]$	ϕ $\left[\frac{\text{pore reaction rate}}{\text{pore diffusion rate}}\right]$	Rate limiting mechanism
$<< 1$, $!\mathscr{D}$	$<< 1$, $!\mathscr{D}$	$<< 1$, $!\mathscr{D}_{\text{pore}}$	kinetics
$<< 1$, $!\mathscr{D}$	$<< 1$, $!\mathscr{D}$	$>> 1$, $!k_{\text{pore}}$	pore diffusion
$<< 1$, $!\mathscr{D}$	$>> 1$, $!k_{\text{surf}}$	$<< 1$, $!\mathscr{D}_{\text{pore}}$	n/a
$<< 1$, $!\mathscr{D}$	$>> 1$, $!k_{\text{surf}}$	$>> 1$, $!k_{\text{pore}}$	pore diffusion
$>> 1$	$<< 1$, $!\mathscr{D}$	$<< 1$, $!\mathscr{D}_{\text{pore}}$	kinetics
$>> 1$	$<< 1$, $!\mathscr{D}$	$>> 1$, $!k_{\text{pore}}$	pore diffusion
$>> 1$	$>> 1$, $!k_{\text{surf}}$	$<< 1$, $!\mathscr{D}_{\text{pore}}$	bulk diffusion
$>> 1$	$>> 1$, $!k_{\text{surf}}$	$>> 1$, $!k_{\text{pore}}$	bulk or pore diffusion

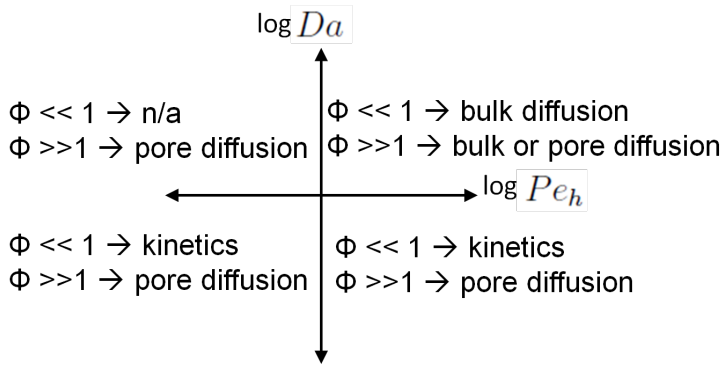


Figure 4.3: Non-dimensional parameters and corresponding rate limiting mechanisms.

4.4 Numerical Model

A numerical finite difference model was also implemented to validate the assumption that four terms in a Fourier series expansion is sufficient to capture the inlet effects. The channel was modeled using 100 grid points in the transverse direction and grid points of variable spacing in the stream-wise direction. A numerical integrator determined the optimal spacing of stream-wise grids. To model each node, Equation

4.1.5 was discretized using a an upwind differencing scheme for the convection on the left-hand-side and a central differencing scheme for the diffusion on the right-hand-side. The interface boundary condition (Equation 4.2.14) was satisfied using a finite difference equation in the transverse direction.

Chapter 5

Model Results and Discussion

This chapter will present and discuss results from the model by itself and results from validating the model by comparison with experimental data. The results that will be presented are species concentration profiles in the stream-wise and transverse directions and species conversion efficiency. The species concentration profile was solved in three ways: the analytical model with one term Fourier series expansion, the analytical model with four term Fourier series expansion, and the numerical model. The species conversion efficiency was solved for a range of varied terms in the Fourier series expansion in the analytical model, and it was also solved using the numerical model.

5.1 Species Concentration

A high resolution plot of the four term solution for species concentration for a flow rate of 500 sccm and temperature of 400 K is shown below in Figure 5.1. For this condition, $Da = 0.010$ and $Pe_h = 20.4$.

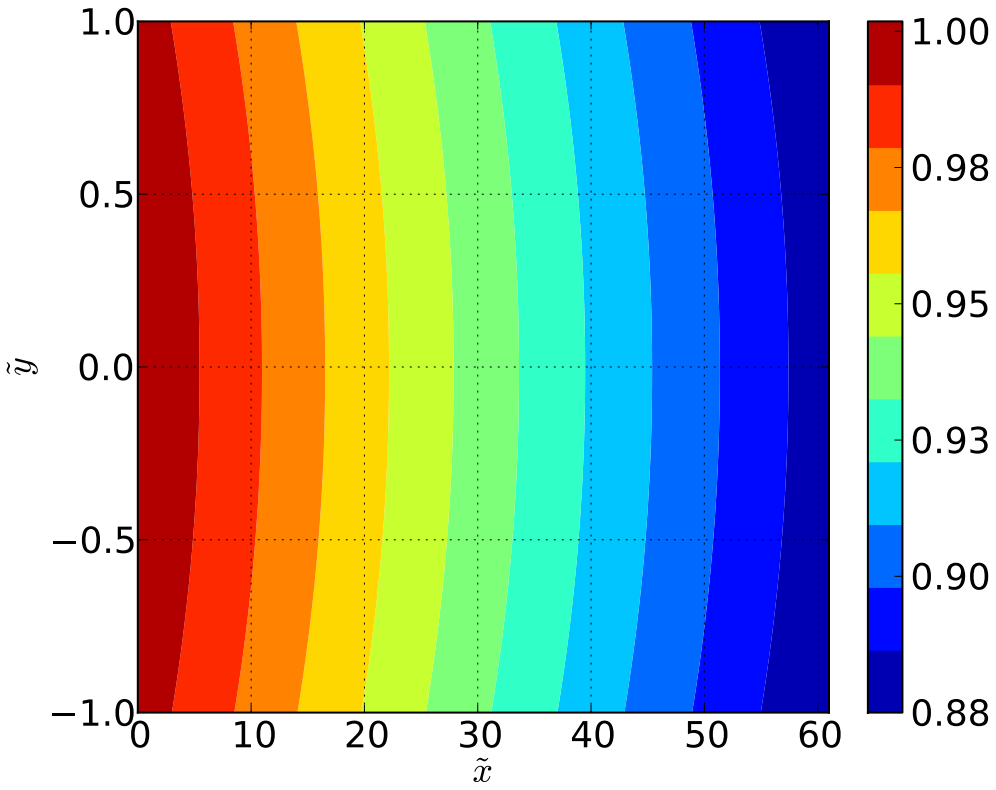


Figure 5.1: Contour plot showing species concentration profile as a function of $\tilde{x}$ and $\tilde{y}$ for four-term Fourier series solution. $\dot{V} = 500$ sccm and $T = 400$ °C.

Note that the species concentration appears to have a parabolic shape in the transverse direction at every location in the stream-wise direction. This indicates that the species concentration profile is fully developed almost immediately which is consistent with the low Pe_h condition. If Pe_h were much larger, the species concentration would appear to be uniform at the inlet, developing a parabolic shape as the flow moves downstream. An example of this in which the Peclet number has been increased by a factor of ~ 25 is shown in Figure 5.2.

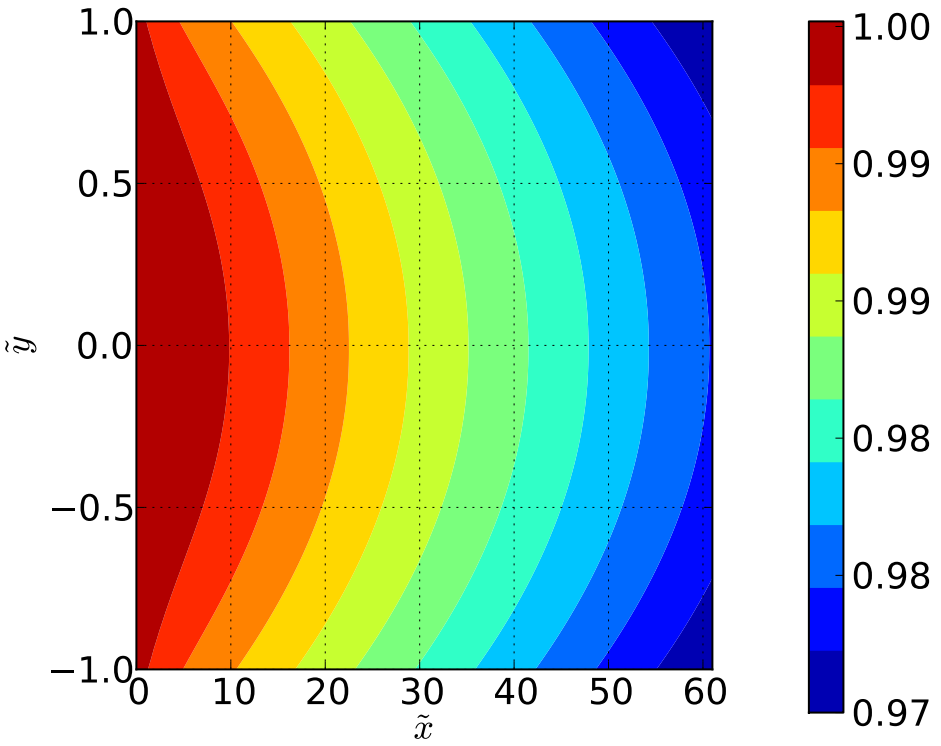
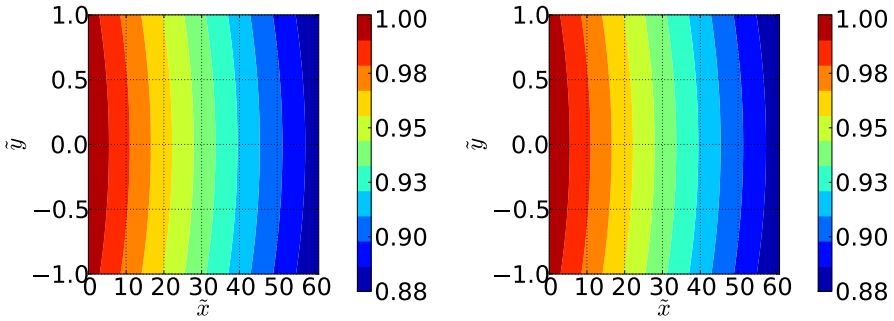


Figure 5.2: Contour plot showing species concentration profile as a function of $\tilde{x}$ and $\tilde{y}$ for four-term Fourier series solution with high Peclet number. $Pe_h = 500$ and $Da = 0.010$.

For this condition, the species concentration profile becomes fully developed quickly, but not immediately. There is clearly a non-parabolic profile visible near the entrance. This is because the relatively high Peclet number increases the entrance length.

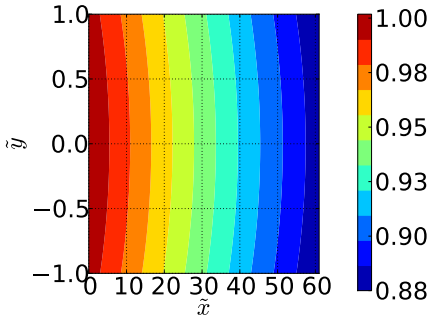
The species concentration model was validated by varying the number of terms in the Fourier series expansion in the analytical model and by comparing the four term analytical result with the numerical model result. Plots comparing the species concentration profiles for one term and four terms in the Fourier series expansion of the analytical model as well as the numerical model are shown in Figures 5.3a, 5.3b,

and 5.3c.



(a) One term.

(b) Four terms.



(c) Numerical.

Figure 5.3: Contour plots showing species concentration profile as a function of $\tilde{x}$ and $\tilde{y}$ for one-term and four-term Fourier series solutions as well as the numerical solution. $\dot{V} = 500$ sccm and $T = 400$ °C.

This shows that for the same flow rate and temperature, the species concentration profile predicted by each technique is very nearly the same. This is because the flow rate (Peclet number) is sufficiently low that the species concentration profile is fully developed almost instantly. A four term solution technique was chosen, somewhat arbitrarily in hindsight, because the eigenvalue solution technique was initially implemented for this number of terms. Also, the computational time was negligibly different from the one term solution, and there was no noticeable improvement in the solution when more than four terms were used. Even for high flow rates for

which entrance effects are important, more than four terms would be unnecessary because additional terms beyond the fourth term are nearly independent of surface Dahmköhler number, and the Fourier coefficients become vanishingly small. The fifth term varied less than one percent over a practical range of Da from 0.01 up to 0.25, and the fifth Fourier coefficient for this condition was $2.22\text{x}10^{-3}$ over the entire range. To provide some sense of how insignificant this is, the value of the first Fourier coefficient was 1.002, and thus, the first term of the Fourier series was dominant in determining the solution.

As yet another means of showing that the analytical model is effective for predicting conversion efficiency, conversion efficiencies were calculated for varied number of terms in the Fourier series expansion, and then the analytical conversion efficiencies were compared to the numerical model conversion efficiency. A plot showing this result for the Fourier series approximations varied from 1 to 10 terms and the numerical model is shown in Figure 5.4.

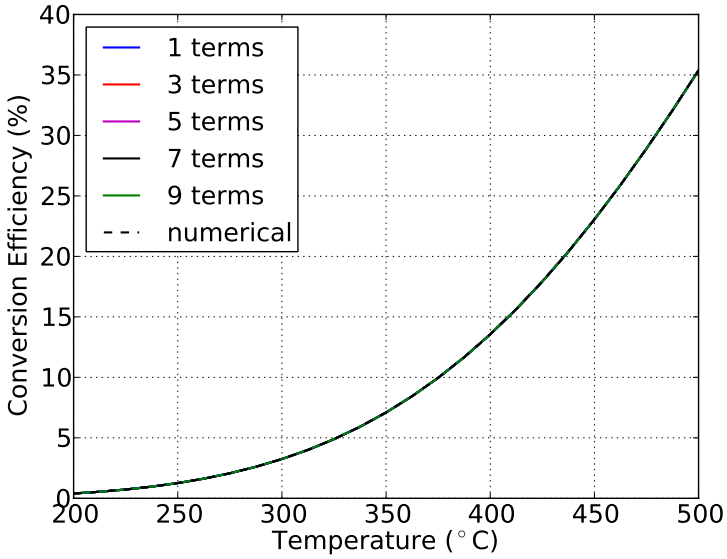


Figure 5.4: Plot showing predicted conversion efficiency for the numerical model and the analytical model using 1 to 10 terms. $\dot{V} = 500$ sccm.

The results shown in Figure 5.4 indicate that the predicted conversion efficiency is almost completely independent of the number of terms used in the Fourier series expansion. Also, there is good agreement between the analytical and numerical models.

5.2 Species Conversion Efficiency

5.2.1 Comparison with Experimental Results

Experimental results for conversion efficiency versus temperature were used to calibrate the kinetic fit parameters in Equation 4.2.21, and the model was used to predict conversion efficiency versus temperature for various flow rates for which experimental data were available. For all results, a four term approximation was used in the Fourier series expansion. Values used for key parameters in the model are shown in Table 5.1. The results are shown in Figure 5.5.

Table 5.1: Values of parameters used in catalyst model.

Parameter	Porosity	Tortuosity	Wash-Coat Thickness	A	T_a
Value	0.97	1.03	5 μm	201.5x10^3 s^{-1}	5,739 K

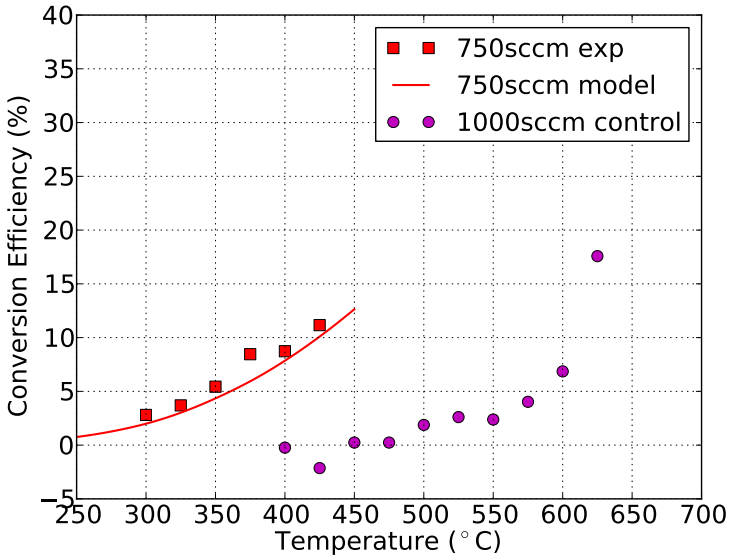


Figure 5.5: Plot of hydrocarbon conversion efficiency versus temperature comparing performance of catalyst with control experiment.

In Figure 5.5, the purpose of the 1000 sccm control experiment was to observe HC conversion not due to catalytic reaction on the nanowire surface. The control experiment consisted of the same rig with the alumina holder, but no catalyst samples were placed inside of the alumina holder. The flow rate of 1000 sccm was chosen because this resulted in approximately the same velocity in the catalyst channel as was the case with catalyst samples in the holder. In the control experiment, all reactions were considered to be bulk reactions or catalytic reactions on the alumina holder.

As shown in Figure 5.5, the fit of the model to the experimental data is quite good through a temperature of 425 °C. The quality of the fit indicates that the

assumption of uniform flow does not introduce any noticeable inaccuracy in matching experimental results. For higher temperatures the model was observed to underpredict the experimental hydrocarbon conversion efficiency. This may have been the result of homogeneous gas-phase reactions in the bulk flow; the control plot shows that these reactions begin to occur at around 500 °C.

A parameterized plot of conversion efficiency versus temperature for four flow rates for experimental and model results is shown in Figure 5.6. Note that the parameters were calibrated at only the 250 sccm condition.

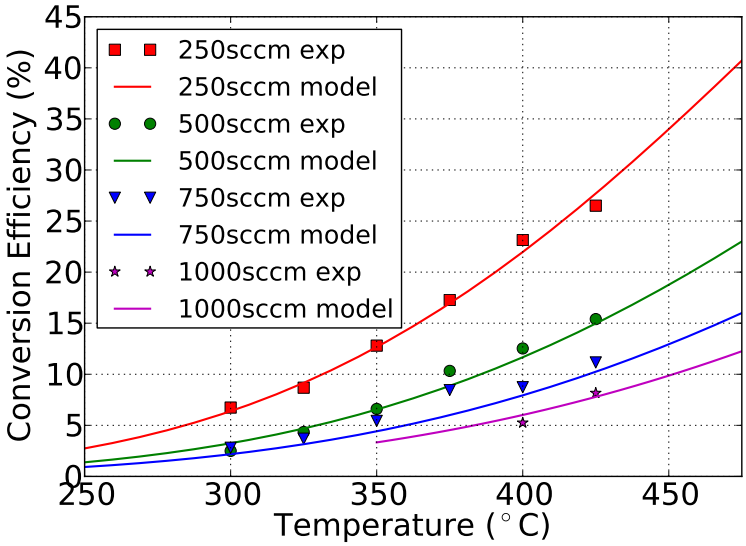


Figure 5.6: Parameterized plot showing hydrocarbon conversion efficiency as a function of temperature for various flow rates for experimental and modeling results.

Figure 5.6 shows good trendwise correlation with the data, illustrating the model's ability to capture the integrated effects of transport even with the limitation of simple one-step kinetics. Its ability to fit the experimental data suggests that with further development for specific applications it may be useful for future catalyst design.

For all results presented here, the Peclet number was greater than one, the Dahmköhler number was near one, and the Thiele modulus was much less than one. This means that conversion within the porous media was limited by kinetics for all conditions, but because the Dahmköhler number was near unity, bulk diffusion and effective kinetics at the interface between the channel and the porous media were both important. Thus, bulk diffusion was important in the overall conversion process. This suggests that the model can be a useful tool for improving catalyst performance in systems where the chemistry is well understood but transport effects have not been thoroughly investigated.

Due to the fact that the Thiele modulus was much less than one, it can be concluded that a substrate material with a higher surface area and a capacity for higher catalyst metal loading, like γ-alumina, can offer a great deal of benefit over ZnO nanowires. This is because higher catalyst metal loading increases the volumetric reaction rate coefficient, k_{pore}. A higher specific surface area can potentially mean a smaller Knudsen length scale, which results in reduced pore diffusion rate, but this effect is not important until chemical kinetics are sufficiently fast that the Thiele modulus is near or greater than one.

In the future, this model might be useful as a means of validating kinetic mechanisms by using the equation relating volumetric reaction rate coefficient in the porous media with reaction rate coefficient on the particle surface (Equation 4.2.22) to back out the local surface kinetics for a particular reactant species and catalyst metal [42].

5.2.2 Predicted Conversion Efficiency versus Nanowire Length and Spacing

To illustrate the utility of this fully coupled species transport and reaction model, the model was used to determine the effect of varying nanowire spacing and nanowire length. This is analogous to varying pore diameter and wash-coat thickness for a traditional alumina catalyst substrate. In the case of nanowires, the nanowire spacing was considered as the length scale for determining the Knudsen number to model the diffusion within the nanowire porous media. It was assumed that these parameters could be varied independently of specific surface area. It was also assumed that particle size and likelihood of agglomeration were unaffected by either nanowire length and/or Knudsen length scale. With these assumptions, Equation 4.2.22 was used to determine how the volumetric reaction rate changed with Pt/Pd particle density.

Nanowire length was varied from 1 to 25 μm for Knudsen lengths (nanowire spacings) of 0.5 nm, 1 nm, 10 nm, and 100 nm for two cases, one where the total amount of Pt/Pd deposited per unit nominal substrate area is held constant and the second where the Pt/Pd particle number density on the surface of each nanowire is held constant. The second case results in an increase in Pt/Pd loading that scales linearly with nanowire length. The reference nanowire length was 5 μm, and the reference Knudsen length scale was 100 nm (same as used in the results shown above). The results for the nanowire scaling are shown in Figures 5.7a and 5.7b. For the first case, it was necessary to scale the pre-exponential kinetics fit parameter by the reference nanowire length of 5 μm divided by the nanowire length. This was based on the assumption that if nanowire length was doubled, the volumetric loading of catalyst metal would be reduced by half for the same total amount of catalyst loading, and therefore, the kinetics fit parameter would be reduced by half also. A mathematical

explanation of this is given in Section 4.2.3 by Equations 4.2.21 and 4.2.22.

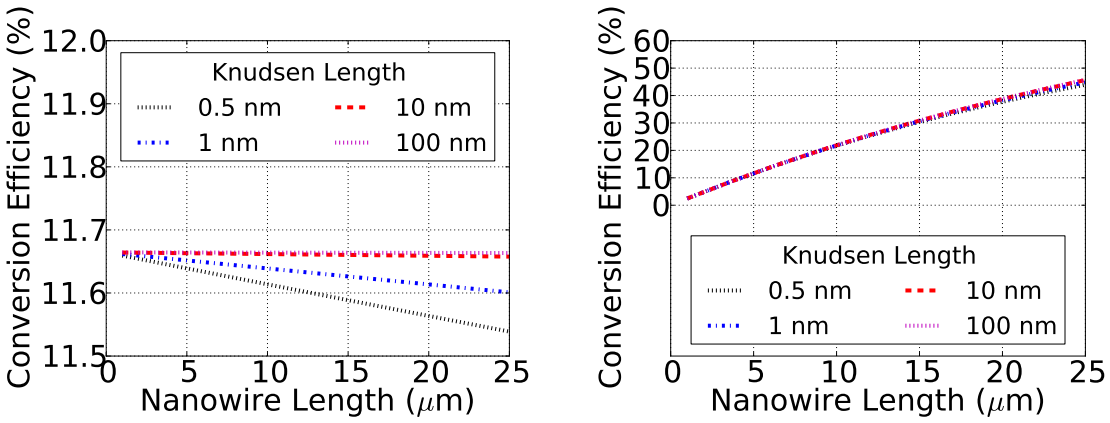


Figure 5.7: Plot of conversion efficiency v. nanowire length for the case of (a) constant total Pt/Pd and (b) constant Pt/Pd particle density. $\dot{V} = 500$ sccm and $T = 400$ °C

The results indicate that, for the conditions considered, decreasing Knudsen diffusion length scale and/or increasing nanowire length have little effect on conversion efficiency. Note the small range of the y-axis in Figure 5.7a. With Pt/Pd density constant, conversion efficiency increases with nanowire length, and this effect is nearly linear. The effect of varied Knudsen length is roughly the same in both plots, but the y-axis scaling changes the apparent magnitude of this effect. The linearity of conversion efficiency with respect to nanowire length in the second case indicates that the conversion efficiency is dominated mostly by the total mass of Pt/Pd on the substrate, rather than substrate characteristics, like Knudsen length scale or substrate thickness, that limit diffusion.

For this condition, Pe_h was 20.4, Da was 0.052, and ϕ was 1.42x10^{-4}. This small value of Dahmköhler number ($Da << 1$) indicates that bulk diffusion was not rate controlling relative to pore diffusion and/or kinetics. The small Thiele modulus

($\phi << 1$) indicates that kinetics were the rate controlling mechanism for this condition. These observations lead to the conclusion that changing the reactor design in such a way as to reduce the Peclet number, thereby increasing residence time, (for example, by using multiple channels with a smaller gap height) can have a large impact on improving conversion efficiency (as indicated in Equation 4.2.4), and increasing Pt/Pd loading can have a stronger effect on conversion efficiency than manipulating the pore transport properties. Interestingly, increasing specific surface area, while it can reduce pore diffusion due to higher tortuosity and smaller pore size, probably improves performance in that Pt/Pd loading can be increased.

5.2.3 Optimization of Channel Height

In the experiments the species conversion efficiencies were limited by both kinetics and bulk diffusion. As an example of the model's potential usefulness for catalyst design, it was used to predict how diffusion effects vary with channel geometry and to determine the channel geometry that maximizes species conversion per unit volume, i.e. species conversion density, for a catalytic reactor of fixed catalyst surface area. This optimization strategy minimizes package size of a catalytic reactor for a given amount of catalyst metal and support material.

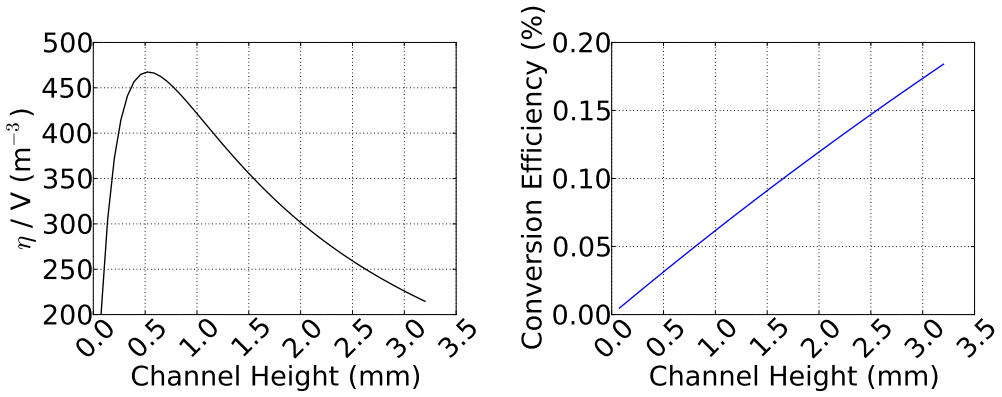
Bejan [43], da Silva, and others [44, 45, 46] have developed models and optimization techniques that are highly effective for maximizing compact heat exchanger performance for a given package space and pumping power requirement. Given the analogy between heat and mass transfer, these optimization techniques are relevant for catalytic reactors. The heat transfer analysis of Bejan [43] was adapted to species transport and used here to optimize the species conversion density of the catalytic reactor.

A catalytic reactor consisting of multiple stacked parallel plates covered with

Pt/Pd-coated ZnO nanowires, a design geometrically similar to the experiments, was modeled. The plate length and distance between plates were varied while maintaining the same total catalyst surface area and reactor cross sectional area. There was thus a trade-off between plate length and the number of plates. The baseline plate spacing was 2.5 mm with an arbitrary number of plates, each with a baseline length of 76.2 mm. The baseline flow velocity through the reactor was 16.7 cm/s; this would be for infinitesimal plate thickness and would increase with increasing number of finite thickness plates due to the smaller flow area within the given cross sectional area of the reactor. The plates were assumed to each have thickness of 550 μm. The 16.7 cm/s velocity corresponds to 500 sccm for the 2.5 mm plate spacing. The velocity was calculated using

$$U = U_0 \frac{h_0}{h_0 + t} \frac{h + t}{h} \tag{5.2.1}$$

where the subscript 0 indicates the baseline value, t is the plate thickness, and U and h have been previously defined. As h approaches zero, velocity approaches infinity. The temperature was 400 °C. Plots of conversion efficiency per unit volume and conversion efficiency versus channel height between adjacent catalyst plates are shown in Figure 5.8.



(a) (b)

Figure 5.8: Plot of (a) conversion efficiency per unit volume and (b) conversion efficiency versus channel height. Channel height is the vertical distance between adjacent catalyst plates. The nominal flow velocity was 16.7 cm/s and the temperature was 400 °C.

This shows that the optimum conversion efficiency density occurs at a channel height of 520 μm. For channel height less than 520 μm, Pe_h increases asymptotically with decreasing channel height. This is because the flow volume approaches zero as channel height approaches zero, causing an asymptotic increase in velocity and space velocity (inverse of residence time). Consequently, the residence time becomes too short relative to the transverse diffusion time to achieve high conversion rates. For channel height greater than 520 μm, Pe_h decreases with increasing channel height and Da increases linearly. As channel height increases beyond 520 μm, species conversion efficiency increases, but because the volume of the reactor increases faster, the species conversion density decreases. A more useful application of this optimization strategy should be employed to account for pressure drop effects, but this is beyond the scope of this work.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

An analytical model was developed to demonstrate that a physics-based, analytical transport sub-model along with an empirical chemical kinetics sub-model can predict catalyst performance remarkably well. ZnO nanowires were used as a support material in fundamental experiments that were used to calibrate and validate the model. The model showed that separation of variables with a spline fit and numerical solver to determine the eigenvalues can effectively be used to fit experimental data using empirically-determined one step Arrhenius kinetics. The key contribution of this model is a systematic means of determining the eigenvalues needed to solve the model for continuous values of chemical kinetics and arbitrary flow rates. After fitting the chemical kinetics parameters to a single set of experimental results, the agreement between the model and the remaining sets of experimental results demonstrated that a simple analytical model can capture the essential physics of the catalyst system remarkably well. This was further validated by comparing the analytical model to a finite difference model. The quality of fit for the experimental data indicates that the assumption of a uniform flow profile, while not physically correct, did not noticeably affect the accuracy of the model.

The model results implied that the hydrocarbon conversion efficiency in the experimental results was limited by both bulk diffusion and chemical kinetics. The model provides a straightforward way of predicting conversion efficiency and species

concentration profile as a function of channel height, channel length, and wash-coat thickness or nanowire length when catalyst particle size and catalyst particle surface kinetics are fixed for a catalytic reactor with known flow rate and temperature operating conditions. In addition, the model clarifies the relationship between important non-dimensional parameters such as Peclet number, Dahmköhler number, and Thiele modulus. The model has shown that maximum species conversion density for fixed catalyst surface area in a reactor with geometry similar to the experiments consisting of stacked parallel plates occurs at a channel height of 521 μm.

ZnO nanowire substrate experimental results were also compared to results for traditional γ-alumina-coated and uncoated cordierite substrates. While the nanowires had only specific surface area of about 7 $\frac{\mathrm{m}^2}{\mathrm{g}}$, which is quite small compared to the industry standard of around 150 $\frac{\mathrm{m}^2}{\mathrm{g}}$ for γ-alumina, the nanowires performed nearly as well as the γ-alumina-coated substrates when compared to the uncoated substrates. This can likely be explained by the partially ballistic nature of sputter coating, which resulted in coating only the first few microns of depth into the γ-alumina washcoat, which was several microns in thickness. This meant that the large surface area of the traditional substrate was largely underutilized. This is supported by the model results because the Thiele modulus for the ZnO nanowires was much less than one, indicating that pore diffusion was not a rate limiting effect.

6.2 Future Work

For the most part, this research came to a cohesive conclusion. However, the model generated several ideas that could offer substantial improvement to the way catalysts are designed.

The model should be compared to experimental results for material systems

that span a range of Thiele modulus from much less than one (e.g. ZnO nanowires) to much greater than one (e.g. a traditional γ-alumina substrate with high catalyst metal loading and small, tortuous pores). This would validate the model as a tool for predicting which mechanism - pore diffusion transport or volumetric chemical kinetics - dominates the rate of hydrocarbon oxidation in a catalyst.

The model should also be used to predict the hydrocarbon conversion performance of several catalysts with different channel geometries that are otherwise identical. This would validate the relative importance of convective transport, residence time, bulk diffusion, and effective interface kinetics.

It would provide a great deal of useful insight if this model were compared to the model presented by Joshi *et al.* [32, 31, 30]. This would provide a means of comparing the relative inaccuracy introduced by the uniform flow assumption of this model with the inaccuracy introduced by the 1D assumption of the Joshi model. It is likely that the Joshi model will not hold up for high flow velocity conditions where the entrance length is much higher and a plug flow assumption is more reasonable. The model presented here has been shown to work well for laminar flow, despite the uniform flow assumption. It is likely that this assumption would result in even less error at higher flow velocities.

Developing a means of accounting for pressure drop in channel geometry optimization with the model would provide a great deal of insight into catalyst design strategy. As emissions regulations continue to tighten, it will be imperative that more researchers bridge the gap between strict catalysis/after-treatment research and strict in-cylinder combustion research because system level optimization will become more important. As a step toward this goal, it would be a good idea to couple the analytical catalyst model with an engine model that accounts for back pressure effects.

Without doing this, as channel height in the catalyst is decreased, the conversion efficiency increases more rapidly than the pressure drop increases so there is no way to optimize catalyst geometry that accounts for pressure drop. With even a simple engine model, the penalty for pressure drop would grow more rapidly with increasing pressure drop because the engine would need to burn more fuel (and produce more emissions) to maintain the same power output. This provides a natural metric for improving catalyst geometry in a way that accounts for pressure drop. From what I have seen in the catalyst literature, this has not been done with the sophistication and simplicity that has been presented in heat transfer literature for similar optimization. Given the analogy between heat and mass transfer, this is a logical knowledge gap to fill.

Part II

Model and Experimental Results for a Heat Exchanger with Thermoelectric Devices for Waste Energy Recovery from Diesel Engine Exhaust

Chapter 7

Introduction and Background

7.1 Motivation

One third or more of the energy content of the fuel of a typical diesel engine is expelled through the tailpipe as waste heat, so any amount of waste heat usefully recovered is a benefit to vehicle efficiency [47, 48, 49]. Potential waste heat recovery technologies currently under consideration for automotive applications are organic Rankine cycles [50], turbo-compounding [51], direct usage of the waste heat as thermal energy, powering absorption chillers [52], and thermoelectric (TE) devices [50, 53, 54, 55, 56, 57, 58]. Thermoelectric devices in particular are appealing for automotive applications in that they do not have moving parts, and thermal energy can be converted into electricity directly. Their high cost and low efficiency are currently issues, but the potential use of relatively inexpensive and abundant element materials such as $Mg_2Si_{0.5}Sn_{0.5}$ and $MnSi_{1.75}$ for TE devices shows promise for future cost reductions and improved performance [59]. A schematic of a TE leg pair, which is the most fundamental part of a TE device (a device typically consists of hundreds of leg pairs) is shown in Figure 7.1.

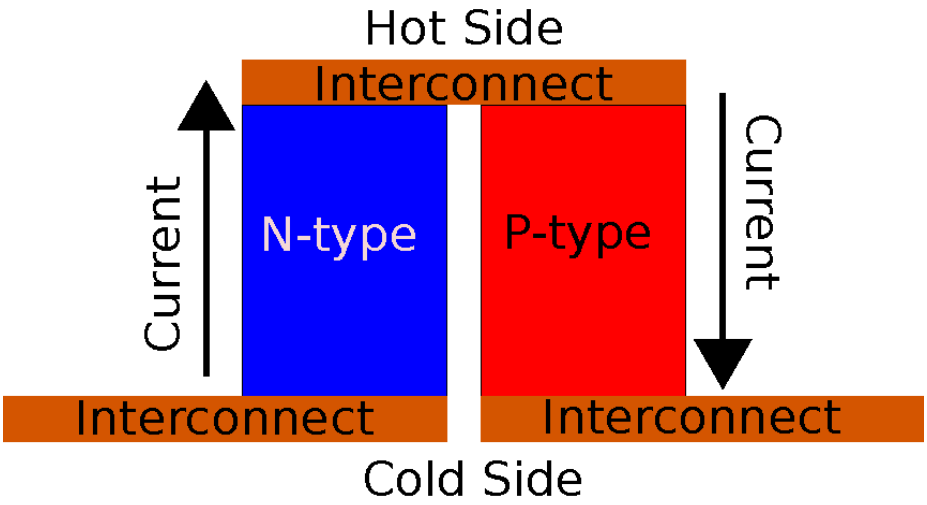


Figure 7.1: Schematic of TE leg pair. A TE leg pair is the smallest unit that can function as a standalone TE device. A typical TE device consists of hundreds of these pairs in series and/or parallel arrangement.

7.2 Literature Review

A number of researchers have reported both modeling and experimental work with TE vehicle waste heat recovery with varying emphasis on heat exchanger performance and TE device performance. Stobart and coworkers modeled and experimentally tested TE performance for devices with exhaust temperatures up to 800 K. Because of internal thermoelectric conversion, there is a difference between the hot side and cold side heat transfer of TE devices. This heat transfer asymmetry was not accounted for in the work by Stobart and coworkers. They used an efficiency model based on the average figure of merit (ZT) of the TE material that assumes optimal device geometry and optimal current [60, 61]. Optimal geometry consists of n-/p-type leg area ratio, leg height, the distance between adjacent legs. Modeling efforts by Hendricks *et al.* [54, 53] reported using temperature-dependent TE properties (Seebeck coefficient, electrical resistivity, and thermal conductivity) to determine optimal TE leg areas, lengths, and device designs. Miller *et al.* studied heat trans-

fer and heat exchanger optimization for a combined TE and organic Rankine cycle waste heat recovery system. Miller *et al.* calculated the TE device efficiency based on an average ZT for state of the art TE materials evaluated at typical operating temperatures [50]. Hussain *et al.* developed a model with single node TE devices that accounted for transient behavior and thermal asymmetry. In their model, the spatial variation and temperature dependence of the TE properties in individual TE devices were not accounted for. No attempt was made to account for restriction of the exhaust flow in the form of back pressure on the engine [55]. Back pressure is an important design consideration because the pumping effort the engine must exert to expel exhaust gases is increased if back pressure in the exhaust system is increased. Increased pumping effort takes useful power away from the engine crank shaft.

Crane and coworkers developed a model for cross-flow and counter-flow heat exchanger configurations. They used an analytical, thermally-lumped TE leg performance model that correctly accounted for thermal asymmetry. Some of this modeling work was also validated with experimental results. Crane's most recent model incorporated transient performance [58, 57, 56]. None of the previous work discussed here used a TE model that accounted for spatial- and temperature-variant properties within the TE material of an individual TE couple.

7.3 Approach

This dissertation reports a modeling study of TE vehicle waste heat recovery devices based on $Mg_2Si_{0.5}Sn_{0.5}$ and $MnSi_{1.75}$ materials with consideration of both system level heat exchanger performance and TE device performance. The model presented in this dissertation offers an improvement over existing models because it used a finite difference method to account for spatial- and temperature-variant TE

properties at the individual TE leg pair scale. The model also coupled the hot- and cold-side convection heat flux to the TE heat flux, thus accounting for the thermal asymmetry, using a numerical root finding algorithm. This model incorporated back pressure as part of an optimization metric in the form of pumping power requirement. The output of the system model was heat transfer, TE power, pumping power requirement, and the net power output, where the net power output is the total TE power minus the pumping power required to move the coolant and exhaust through the heat exchanger. The model was used to predict overall performance of several heat exchanger systems, incorporating different fin geometries and flow arrangements for enhancing heat transfer and TE power generation. The important TE parameters were optimized using a numerical algorithm, and the optimization metric used by the algorithm was the net power of the system.

This work also includes two generations of TE waste heat recovery systems that were built and tested in the exhaust system of a Cummins 6.7 L turbo Diesel engine. The experimental work was used to validate the model so that the model could be used both as a design tool and a means of thoroughly optimizing various parameters in the TE waste heat recovery system design space.

The first generation TE system consisted of a compact heat exchanger without any TE devices. This system was built to validate the convection heat transfer and flow model using only a small portion of the exhaust flow of the Cummins engine. In order to validate the model at full scale, a second generation TE heat exchanger system was constructed to utilize the entire flow of the Cummins engine. Experimental apparatus, methods, and results from each heat exchanger will be presented separately in Chapters 10 and 11.

A key contribution of this work was recognizing that for almost any practical

TE device application, the boundary conditions on the hot and cold side will be convection boundary conditions rather than specified temperature boundary conditions. The TE model developed here accounts for that, and in addition, the model accounts for spatially/thermally variant properties in the direction of heat flow and thermal asymmetry.

Chapter 8

Model Approach

8.1 Model History

Initially, the heat exchanger was modeled using the traditional number-of-transfer-units (ϵ-NTU) method used for standard heat exchangers. The thermal effects of the TE devices were lumped into the overall heat transfer coefficient by assuming pure conduction in the TE devices, and the output was determined by calculating the efficiency using [62]:

$$\eta = \frac{T_h - T_c}{T_h} \frac{\sqrt{1+ZT}-1}{\sqrt{1+ZT}-T_c/T_h} \tag{8.1.1}$$

where η is the thermal efficiency of the device, T_h is the hot side temperature of the device, T_c is the cold side temperature of the device, and ZT is the dimensionless figure of merit evaluated at the mean temperature. ZT is a function of the Seebeck coefficient, the electrical resistivity, and the thermal conductivity; it is defined as

$$ZT \equiv \frac{\alpha^2 T}{\rho k} \tag{8.1.2}$$

where α is the Seebeck coefficient, T is the mean of the hot- and cold-side temperature of the device, ρ is the electrical resistivity, and k is the thermal conductivity. For Equation 8.1.1, all properties are evaluated at T.

One shortcoming of this method is that TE properties are typically highly dependent on temperature and are therefore spatially variant along the length of a TE device. In consideration of this shortcoming, a numerical, finite difference model

was developed for more accurately modeling the TE performance, and this quickly led to the realization that the thermal asymmetry created by the energy conversion in the TE devices must be accounted for in the heat exchanger model. This rendered traditional heat exchanger modeling methods ineffective. The resulting solution to this complication will be explained in the subsequent sections.

8.2 Model Methods

8.2.1 Thermoelectric Device Model

The TE devices were modeled using Domenicali's energy balance equation and the thermoelectric heat flux equation [63, 64]:

$$\frac{d}{dx}\left(k\frac{dT}{dx}\right) = -\rho J^2 + JT\frac{d\alpha}{dx} \tag{8.2.1}$$

and

$$q = JT\alpha - k\frac{dT}{dx} \tag{8.2.2}$$

where x is the coordinate along the direction of heat and current flux, k is the thermal conductivity, T is the temperature, ρ is the electrical resistivity, J is the current flux, and α is the Seebeck coefficient, all evaluated at T.

For completeness, the entire solution for Equations 8.2.1 and 8.2.2 will be presented here. This solution has been presented in the literature [64] with a critical sign error, and the present work corrects this error.

A tractable solution can be achieved through several steps of algebraic manipulation. First, rearranging Equation 8.2.2 yields

$$\frac{dT}{dx} = \frac{JT\alpha - q}{k} \tag{8.2.3}$$

This is the final equation for determining the temperature gradient. In order to determine the heat flux gradient, Equation 8.2.3 is multiplied by k and substituted

into Equation 8.2.1, resulting in

$$\frac{d}{dx}\left(JT\alpha - q\right) = -\rho J^2 + JT\frac{d\alpha}{dx} \tag{8.2.4}$$

For constant current density (e.g. if the cross-sectional area of the leg is constant), the product rule can be used to expand the left-hand-side to

$$J\left(\frac{dT}{dx}\alpha + \frac{d\alpha}{dx}T\right) - \frac{dq}{dx} = -\rho J^2 + JT\frac{d\alpha}{dx} \tag{8.2.5}$$

The temperature gradient in first term on the left-hand-side of Equation 8.2.5 can be replaced by substituting Equation 8.2.3, and after some rearranging, this yields

$$\frac{dq}{dx} = \rho J^2 + J\alpha\frac{JT\alpha - q}{k} \tag{8.2.6}$$

Multiplying the second term ($J\alpha\frac{JT\alpha-q}{k}$) by $\frac{\rho}{\rho}$ and recognizing the definition of ZT (Equation 8.1.2) yields the final result:

$$\frac{dq}{dx} = \rho J^2\left(1 + ZT\right) - \frac{J\alpha q}{k} \tag{8.2.7}$$

Current density, rather than load resistance, is specified in the model. However, it may be desired to specify load resistance for experimental or design purposes. To fully close the model for a specified load resistance, an expression is needed to relate current density to load resistance. This is given by

$$R_{\mathrm{L}} = \frac{V_{\mathrm{S,n}} - I/A_{\mathrm{n}}\rho_{\mathrm{n}}l + V_{\mathrm{S,p}} - I/A_{\mathrm{p}}\rho_{\mathrm{p}}l}{I} \tag{8.2.8}$$

where R_{L} is the load electrical resistance, l is the length of the TE legs, subscripts n and p indicate n- or p-type leg, and V_{S} is the Seebeck voltage, given by

$$V_{\mathrm{S}} = \sum_{i=1}^{N}\alpha_i\left(T_i - T_{i-1}\right) \tag{8.2.9}$$

and the current, I, is given by

$$I = JA \tag{8.2.10}$$

where J is current density of either leg, and A is the cross-sectional area of either leg. Either leg can be used for this calculation, provided the same leg is used for both the current density and the area. The total power was calculated by

$$\dot{W}_{\text{elec}} = IR_{\text{L}} \tag{8.2.11}$$

where $\dot{W}_{\text{elec}}$ is the electrical power output.

For a pair of thermoelectric legs with convection heat transfer on both sides and isothermal interconnects with no generation, the boundary conditions are combined heat flux and temperature. The interconnects, which are typically copper, are assumed to be isothermal because the thermal conductivity is high. Generation due to Joule heating is assumed to be negligible due to high electrical conductivity, and thermoelectric energy conversion at the interface of the interconnects and the TE materials is neglected. A schematic that will be referenced for both the analytical and numerical finite difference equations is shown in Figure 8.2. On both the hot and cold side of the TE leg pair, the temperature of each leg must be equal to the temperature of the isothermal interconnect interface, and this is also the temperature driving convection heat flux. The composite heat flux is given by

$$q_{\text{comp}} = \frac{A_{\text{n}} q_{\text{n}} + A_{\text{p}} q_{\text{p}}}{A_{\text{n}} + A_{\text{p}} + A_{\text{void}}} \tag{8.2.12}$$

where A is the cross-sectional area of the n- or p-type leg or that of the void space, and q is the composite heat flux or the heat flux in the specified leg. Subscripts n, p, void, and comp indicate whether the variable references the n-type, p-type, void space, or composite value. A visual representation of the p-type, n-type and void areas is shown in Figure 8.1. The void area is the empty space between adjacent TE legs, and it is assumed to be perfectly insulated.

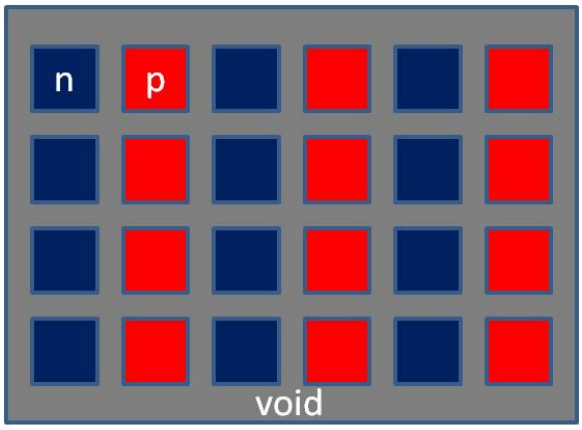


Figure 8.1: Visual reprentation of p-type, n-type, and void areas.

The composite heat flux must be equal to the convection heat flux on both the hot and cold sides, given by

$$q_{\mathrm{h,comp}} = U_{\mathrm{h}} \left(T_{\mathrm{h,conv}} - T_{\mathrm{h}} \right) \tag{8.2.13}$$

for the hot side and

$$q_{\mathrm{c,comp}} = U_{\mathrm{c}} \left(T_{\mathrm{c,conv}} - T_{\mathrm{c}} \right) \tag{8.2.14}$$

for the cold side, where U is the overall heat transfer coefficient for the hot or cold side (including thermal resistance due to conduction and contact resistances associated with the Al plate, substrate, and interconnect shown in Figure 8.2 as well as thermal resistance due to convection; see Section 8.2.4), T_{conv} is the temperature of the hot or cold fluid, T is the temperature of the hot or cold side of the TE leg pair, and q_{comp} is the composite heat flux on the hot or cold side of the TE device, given in Equation 8.2.12. Subscripts $_h$ and $_c$ indicate whether the variable corresponds to the hot or cold side. The methods used to obtain U are described in Sections 8.2.3 and 8.2.4. The temperature boundary conditions are specified as follows:

$$T_{\mathrm{n,h}} = T_{\mathrm{p,h}} = T_{\mathrm{h}} \tag{8.2.15}$$

$$T_{\mathrm{n,c}} = T_{\mathrm{p,c}} = T_{\mathrm{c}} \tag{8.2.16}$$

where, as before, subscripts n and p indicate n- or p-type leg, and subscripts c and h indicate cold or hot side.

8.2.2 Thermoelectric Finite Difference Model

Following the procedure from Hogan and Shih [64], the results from Section 8.2.1 can be discretized to the following pair of first order, reverse-looking finite difference equations:

$$T_i = T_{i-1} + \left(\frac{JT_{i-1}\alpha_{i-1} - q_{i-1}}{k_{i-1}} \right) \Delta x \tag{8.2.17}$$

from Equation 8.2.3, and

$$q_i = q_{i-1} + \left(\rho_{i-1} J^2 + (1 + ZT_{i-1}) - \frac{J\alpha_{i-1} q_{i-1}}{k_{i-1}} \right) \Delta x \tag{8.2.18}$$

from Equation 8.2.7, where the subscripts $_i$ and $_{i-1}$ indicate the current and previous nodes, respectively, and the TE properties are all evaluated at the temperature of the previous node. A schematic of the system being discretized is shown in Figure 8.2.

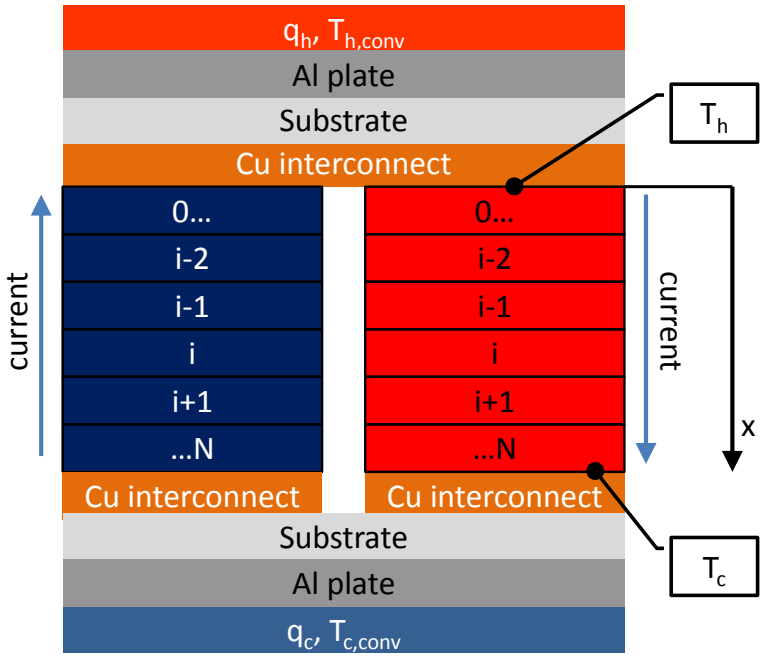


Figure 8.2: Schematic of numerical scheme used to solve TE leg pair. Blue leg is n-type and red leg is p-type.

The algorithm used to solve these coupled finite difference equations deviated substantially from the method described by Hogan and Shih [64] in that the algorithm accounted for a convection (Neumann) boundary condition rather than a temperature (Dirichlet) boundary condition. This is important because in most practical applications will have temperature-driven heat flux, or convection, boundary conditions. The necessary boundary conditions are fluid temperature and heat transfer coefficient on both the hot and cold side of the TE device as well as the hot and cold side temperature of the TE device. The hot and cold side TE device temperature boundary conditions are satisfied by

$$T_{0,\mathrm{n}} = T_{0,\mathrm{p}} \tag{8.2.19}$$

and

$$T_{N,\mathrm{n}} = T_{N,\mathrm{p}} \tag{8.2.20}$$

where subscript 0 indicates the node in contact with the hot side copper interconnect, subscript N indicates the node in contact with the cold side copper interconnect, and the second subscript indicates n-type or p-type TE leg. The heat flux boundary conditions are given by following conditions:

$$U_{\mathrm{h}} \left(T_{\mathrm{h,conv}} - T_{\mathrm{h}} \right) = q_{\mathrm{h,comp}} \tag{8.2.21}$$

and

$$U_{\mathrm{c}} \left(T_{\mathrm{c,conv}} - T_{\mathrm{c}} \right) = q_{\mathrm{c,comp}} \tag{8.2.22}$$

The algorithm proceeds as follows:

1. Estimate hot side heat flux for the n-type leg, hot side heat flux for the p-type leg, and hot side temperature based on pure conduction and convection.

2. Solve for the temperature and heat flux profiles in both the n-type and p-type TE legs using Equations 8.2.17 and 8.2.18.

3. Determine the error between the composite TE heat flux and the convection heat flux (Equations 8.2.21 and 8.2.22) for both the hot and cold sides.

4. Determine the error between the cold side p-type and n-type temperatures, as given by Equations 8.2.19 and 8.2.20.

5. Iterate until the error in steps 3 and 4 is zero.

The iteration was performed by the *fsolve* function of the open source Python 2.7 SciPy package.

8.2.3 Convection Model

Convection heat transfer and pumping power were modeled using standard empirical correlations for turbulent flow. The flow properties (density, dynamic viscosity, thermal conductivity, specific heat) were calculated as a function of temperature using a methodology that is described in detail in Appendix A. The following correlations [65] for Nusselt number based on hydraulic diameter (Nu_D) and friction factor (f) were used [65]:

$$Nu_D = \frac{8.24}{4.36} 0.023 Re_D^{4/5} Pr^{1/3} \tag{8.2.23}$$

for turbulent flow ($Re_D > 2300$) with heat transfer on both sides,

$$Nu_D = \frac{5.39}{4.36} 0.023 Re_D^{4/5} Pr^{1/3} \tag{8.2.24}$$

for turbulent flow with heat transfer on one side and one adiabatic side,

$$Nu_D = 7.54 \tag{8.2.25}$$

for laminar flow ($Re_D < 2300$) with heat transfer on both sides,

$$Nu_D = 5.39 \tag{8.2.26}$$

for laminar flow with heat transfer on one side and one adiabatic side,

$$f = \frac{24}{16} 0.078 Re_D^{-1/4} \tag{8.2.27}$$

for turbulent flow, and

$$f = \frac{24}{Re_D} \tag{8.2.28}$$

for laminar flow, where Re_D is the Reynolds number based on hydraulic diameter, and Pr is the Prandtl number. The leading coefficients in Equations 8.2.23, 8.2.24, and 8.2.27 were scaling factors to account for the fact that the original correlation was

intended for round tubes with circumferentially uniform heat transfer. These scaling factors were based on Table 3.2 of Bejan's Convection Heat Transfer [65].

The coefficient of convection was calculated using [66]

$$h = \frac{Nu_D\, k}{D} \tag{8.2.29}$$

where Nu_D was calculated from Equation 8.2.23, 8.2.24, 8.2.25, or 8.2.26, as appropriate, k is the thermal conductivity of the fluid, and D is the hydraulic diameter of the flow channel, calculated by

$$D = \frac{4A_c}{P} \tag{8.2.30}$$

where A_c is the flow channel cross-sectional area and P is the flow channel wetted perimeter.

The pressure drop per unit length was calculated using [65]

$$\frac{\Delta P}{\Delta x} = f\frac{P}{A_c}\left(1/2\rho U^2\right) \tag{8.2.31}$$

where U was the mean fluid velocity.

A first order Euler method was used to iterate along the stream wise direction of the heat exchanger to calculate temperature and pressure in each subsequent stream-wise node. This will be discussed in Section 8.2.5.

8.2.4 Parasitic Losses

All of the parasitic losses are lumped into the overall heat transfer coefficient, U, in Equations 8.2.21 and 8.2.22. These losses include the following (refer to Figure 8.2):

- convection resistance between the hot or cold fluid and the surface

- thermal resistance of the aluminum plate
- thermal contact resistance between the aluminum plate and the TE substrate material
- thermal resistance of the TE substrate
- thermal contact resistance between the TE substrate and the TE interconnect
- thermal resistance of the TE interconnect
- thermal contact resistance between the TE interconnect material and the TE leg

The aluminum plates, substrates, and interconnects are all identified in Figure 8.2. The overall heat transfer coefficients are found using the following equations:

$$U_h = (R_{conv} + R_1 + R_{1-2} + R_2 + R_{2-3} + R_3 + R_{3-TE})^{-1} \tag{8.2.32}$$

and

$$U_c = (R_{conv} + R_1 + R_{1-2} + R_2 + R_{2-3} + R_3 + R_{3-TE})^{-1} \tag{8.2.33}$$

where R_{conv} is the thermal resistance of the convection (inverse of convection coefficient), R_1 is the thermal resistance of the aluminum plate, R_{1-2} is the thermal contact resistance of the interface between the aluminum plate and the substrate, R_2 is the thermal resistance of the substrate, R_{2-3} is the thermal contact resistance of the interface of the substrate and the interconnect, R_3 is the thermal resistance of the interconnect, and R_{3-TE} is the thermal contact resistance of the interface of the interconnect and the TE leg. The thermal resistance of the aluminum plate (R_1) was modeled based on pure conduction with an assumed thermal conductivity of 200 $\mathrm{W{\cdot}m^{-1}{\cdot}K^{-1}}$ [66]. The thermal resistance of the substrate (R_2) was assumed to be

$5x10^{-3}$ m^2K/kW based on an assumed thickness of 1 mm and a thermal conductivity of 200 W/(m-K) [67, 59]. The thermal resistance of the copper interconnect (R_3) was assumed to be $7.5x10^{-4}$ m^2K/kW based on an assumed thickness of 0.3 mm and a thermal conductivity of 400 W/(m-K). The contact resistances (R_{1-2}, R_{2-3}, and R_{3-TE}) were assumed to be $3x10^{-5}$ m^2K/kW [68].

8.2.5 System Model

The overall system performance with the inclusion of the silicide TE materials (n-type: $Mg_2Si_{0.5}Sn_{0.5}$ and p-type: $MnSi_{1.75}$) was modeled by integrating the TE device model into the heat exchanger model. Properties taken from Gao et al. [69] and Luo et al. [70], and electrical resistivity, thermal conductivity, and Seebeck coefficient were evaluated at the local temperature of each node of each TE leg element using a polynomial curve fit. Temperature was spatially resolved in both the direction along the leg and the stream-wise direction of the heat exchanger.

The model was discretized into 25 finite nodes along the heat exchanger length to allow for stream-wise variation of fluid and TE properties. This was especially important for the TE properties which are highly dependent on temperature. This method considered a single nodal temperature set in each finite stream-wise node of the heat exchanger and then moved on to the next node in the flow direction of the exhaust. The nodal temperature set consisted of exhaust temperature, temperature in all of the TE finite segments, and coolant temperature.

This method correctly accounted for the fact that the heat loss from the hot flow is greater than the heat transferred to the cold flow, which was not properly treated with a traditional heat exchanger solution technique like the ϵ-NTU method. To iterate the model in the stream-wise direction, a first order finite difference method

was used for both the exhaust and coolant:

$$T_{exh,i} = T_{exh,i-1} + \frac{Q_{exh,i-1}}{C_{exh}} \tag{8.2.34}$$

$$T_{cool,i} = T_{cool,i-1} + \frac{Q_{cool,i-1}}{C_{cool}} \tag{8.2.35}$$

where T_i is the temperature in the present node, T_{i-1} is the temperature in the previous node, Q_{i-1} is the heat transfer rate in the previous node, and C is the capacity flow rate (kW/K) of the coolant or exhaust fluid. The subscript exh indicates exhaust flow properties, and the subscript *cool* indicates coolant flow properties. The plus sign in Equation (8.2.34) is due to the fact that heat transfer from the hot side to the cold side is in the direction of decreasing segment index, i, and for the case of Equation (8.2.35), the heat exchanger being modeled is counter-flow so the stream-wise iterations proceed in the opposite direction of the coolant flow.

Pumping power requirement was calculated in each node by multiplying the flow rate by the pressure drop:

$$\dot{W} = \dot{V}_i \Delta P_i \tag{8.2.36}$$

where $\dot{V}_i$ is the node flow rate, and ΔP_i is the pressure drop in the node. This was based on the assumption that the pressure and temperature were both approximately constant in the stream-wise direction of each node. Pumping power in all the nodes was summed up to find the total pumping power, which did not rely on a constant property assumption over the duct length.

A number of system parameters required optimization to maximize net power. For each configuration, a different n-/p-type leg area ratio, leg length, load resistance, and fill fraction were used because each configuration had a different set of these parameters that resulted in maximum power output. Fill fraction is defined as the area of the TE devices divided by the total area, including the void area. In Figure

8.1, this would be the sum of the red and blue areas divided by the total area. Varying fill fraction changed the area through which heat flux passed, and it also changed the temperature boundary conditions for the TE devices, thus potentially resulting in better efficiency. Decreasing the fill fraction, as examined by Kraemer *et al.* [71] and Baker *et al.* [72], effectively worked as a thermal concentrator to increase the heat-flux through each leg pair and thereby increase the difference between the hot- and cold-side temperature.

The most important design space consisted of leg length, fill fraction, and current. These parameters have interdependent effects on power output. The n-/p-type leg area ratio, leg length, current (as a surrogate for load resistance), and fill fraction were selected for maximum net power output using a downhill simplex optimization algorithm [73]. The algorithm provided a minimization function, and the inverse of net power was used as the metric for minimization. Thus, the appropriate parameters were selected for maximum net power when the inverse of net power was minimized. This algorithm was also applied to fin spacing for both the strip fins and offset-strip fins.

8.3 Heat Transfer Enhancement

Four methods were considered for enhancing exhaust side heat transfer: porous metal foams, impinging jets, offset-strip fins, and strip fins. For all cases modeled, the TEM package volume was 29.5 L, the length was 50.8 cm, the width was 50.8 cm, the exhaust duct height was 6.35 cm, the coolant duct height was 2.54 cm, and the exhaust flow enthalpy flux was 122 kW for an inlet temperature of 800 K and a mass flow rate of 7.9 kg/min. This geometry is consistent the second generation heat exchanger which will be described in Chapter 11. This temperature and flow rate

condition is typical of high load, high rpm engine operation and represents a best case scenario. The following configurations were considered in this model: empty ducts, exhaust duct filled with porous metal foam, exhaust duct lined with impinging manifold, exhaust and coolant ducts with strip fins, and exhaust duct with offset-strip fins and coolant duct with strip fins.

In all cases, the temperature-dependent material properties for the n-type ($Mg_2Si_{0.5}Sn_{0.5}$) and p-type ($MnSi_{1.75}$) materials were calculated from Gao *et al.* [69] and Luo *et al.* [70], respectively.

For all configurations except the porous metal foams, a downhill simplex algorithm was employed to optimize TE leg length, TE leg area ratio, TE fill fraction, and TE electrical current. For some enhancement types, various enhancement parameters were also included in the optimization algorithm. Optimization of coolant heat transfer enhancement was not thoroughly investigated due to the fact that the coefficient of convection for the coolant was much higher than that of the exhaust, and thus changing the exhaust enhancement would have little impact on the overall heat transfer rate.

A schematic showing a side cross-section of the duct arrangement is shown in Figure 8.3. The red rectangle represents the exhaust flow, the blue rectangles represent coolant flow, and the gray rectangles separating the blue and red rectangles represent an array of thermoelectric devices consisting of hundreds of the leg pairs shown in Figure 8.2.

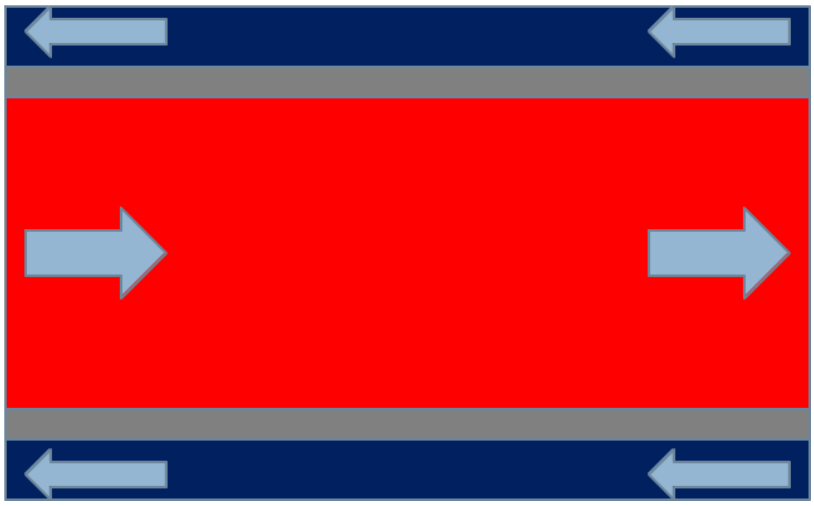


Figure 8.3: Schematic of basic heat exchanger without any convection enhancement.

8.3.1 Porous Metal Foams

Porous metal foams were considered because they increase the effective, or composite, thermal conductivity of the fluid in which they are immersed. The composite thermal conductivity is the mass-weighted conductivity of the fluid and the porous media, and therefore it can be several orders of magnitude higher than the thermal conductivity of a typical gas [65, 74]. Because the coefficient of convection in the coolant was already substantially higher than that of the exhaust, this enhancement was considered for the exhaust side only. The porous metal foams would fill the exhaust duct, they would be bonded to the walls using a brazing process.

The model chosen for pressure drop through the porous media was presented in the literature [74]. The Nusselt number for the heat transfer model was taken from Bejan's Convection Heat Transfer [65] as $Nu_D = 4.93$. The porous media modeled was Duocel aluminum foam with 92% porosity and 10 pores per inch. The effective matrix thermal conductivity was $5.8\frac{W}{mK}$ [75].

The pressure drop was modeled using theory developed by Mancin *et al.*. The

pressure drop was calculated using [74]

$$\Delta P = \frac{2LFG^2}{D_{\text{pore}}\rho} \tag{8.3.1}$$

where L is the duct length in the stream-wise direction, ρ is the fluid density, and expressions for F, G, and D_{pore}, are given in Equations 8.3.5, 8.3.2, and 8.3.3, respectively. The expression for the mass velocity is [74]

$$G = \rho U \tag{8.3.2}$$

where U is the fluid velocity. The expression for hydraulic diameter of the porous channel is [74]

$$D_{\text{pore}} = 0.122 PPI^{-0.849} \tag{8.3.3}$$

where PPI is the number of pores per linear inch (assuming isotropic pore geometry). The expression for Reynolds number based on the hydraulic diameter is [74]

$$Re_D = \frac{D_{\text{pore}}G}{\mu\epsilon} \tag{8.3.4}$$

where μ is the dynamic viscosity of the fluid and ϵ is the porosity of the fluid. The expression for porous media friction factor is [74]

$$F = \frac{1.765 Re_D^{0.1014}\epsilon^2}{PPI^{0.6}} \tag{8.3.5}$$

where all variables have been previously defined.

For porous metal foams, no enhancement was considered on the coolant side.

8.3.2 Impinging Jets

An array of impinging jets was modeled using empirical correlations [76]. A conceptual schematic of the heat exchanger with the impinging jet array is shown in Figure 8.4.

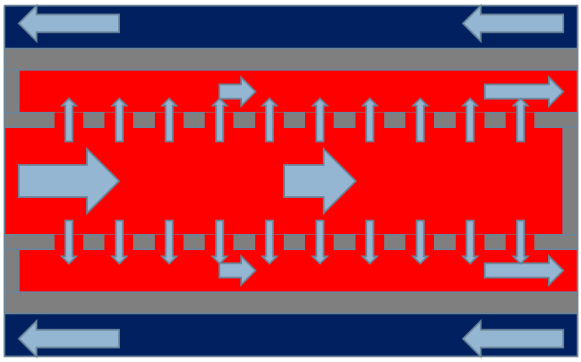


Figure 8.4: Conceptual schematic of heat exchanger with exhaust side impinging jets as heat transfer enhancement.

The correlation for Nusselt number is [76]

$$Nu_D = 0.285 Re_D^{0.710} Pr^{0.33} \left(\frac{H}{D}\right)^{-0.123} \left(\frac{S}{D}\right)^{-0.725} \tag{8.3.6}$$

where Re_D is the Reynolds number based on the jet orifice diameter, Pr is the fluid Prandtl number, H is the distance the between the jet orifice and the plate on which the flow impinges, D is the jet orifice diameter, and S is the distance between adjacent jets. Equation 8.3.6 assumes no cross-flow, and the results will therefore have a high degree of uncertainty because the geometry being used in the heat exchanger (Figure 8.4) will induce cross-flow in both the outer and inner channels.

The pressure drop was modeled using [77]

$$\Delta P = \frac{1}{2} \rho K U^2 \tag{8.3.7}$$

where ρ is the fluid density, K is the minor loss coefficient for a square edged pipe entry (in this case, $K = 0.5$), and U is the fluid velocity going through the orifice. Pressure drop in the stream-wise direction of the primary flow in the inner channel and outer channel was neglected.

There was no optimization of H or S because as each of this was reduced, the net power output increased. These parameters were fixed at H= 1 cm, and S= 1 cm because these dimensions would be easily manufacturable.

8.3.3 Offset-strip Fins

In his book [78], Lee suggests that offset-strip fins offer an exceptionally high ratio of Colburn factor (a dimensionless convection number that relates heat transfer coefficient to mass flow rate) to friction factor. This makes offset-strip fins an ideal candidate for enhancing heat transfer in this application because they can provide a large amount of heat transfer enhancement with minimal increase in pressure drop. Lee provided an example using a model developed by Manglik and Bergles [79], and their model was used in this work. A schematic that illustrates what offset-strip fins are and labels the important geometrical dimensions is shown in Figure 8.5.

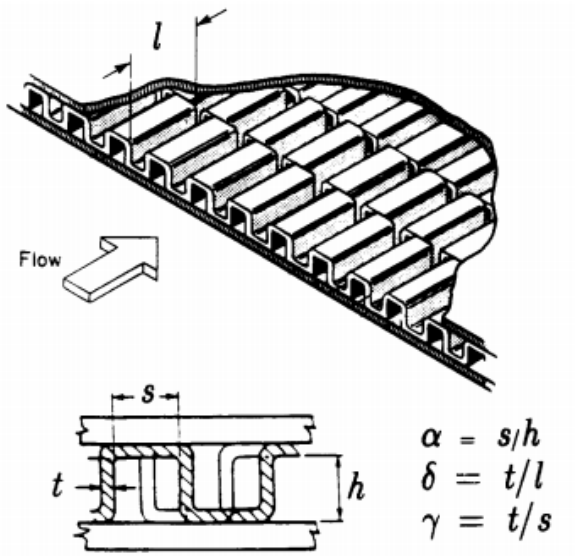


Figure 8.5: Schematic of offset-strip fins sandwiched between two plates. Important dimensions are labeled in the image. Image copied from ref. [79].

Equation 8.2.31 was used to calculate the pressure drop, but the offset-strip

fins required a more complicated formula to determine friction factor. The following equation developed by Manglik and Berlges was used to determine friction factor:

$$f = 9.6243 Re_D^{-0.7422} \alpha^{-0.1856} \delta^{0.3053} \gamma^{-0.2659} \tag{8.3.8}$$

where Re_D was the Reynolds number based on the following hydraulic diameter:

$$d = \frac{4shl}{2\left(sl + hl + th\right) + ts} \tag{8.3.9}$$

where s is the spacing between surfaces of adjacent fins in the same row; h is the vertical height of the duct minus the thickness of the fin; t is the thickness of the fin; and α, δ, and γ are given by Equations 8.3.10, 8.3.11, 8.3.12, respectively. Equation 8.3.8 accounted for minor losses at the duct inlet and outlet. All of these parameters are illustrated in Figure 8.5.

$$\alpha = \frac{s}{h} \tag{8.3.10}$$

$$\delta = \frac{t}{l} \tag{8.3.11}$$

$$\gamma = \frac{t}{s} \tag{8.3.12}$$

Manglik and Bergles also provided an equation for the Colburn factor,

$$\begin{aligned} j = {} & 0.6522 Re_D^{-0.5403} \alpha^{-0.1541} \delta^{0.1499} \gamma^{-0.0678} \times ... \\ & \left(1 + 5.269 \times 10^{-5} Re_D^{1.340} \alpha^{0.504} \delta^{0.456} \gamma^{-1.055}\right)^{0.1} \end{aligned} \tag{8.3.13}$$

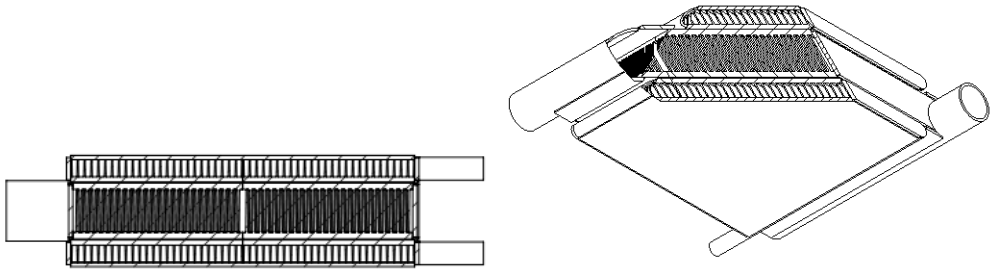
which would then be used to calculate the heat transfer coefficient using

$$h = \frac{j \dot{m} c_p}{A_c Pr^{0.667}} \tag{8.3.14}$$

For the optimization of offset-strip fins, the free parameter was fin spacing, s, and the fin length, l, was held constant at 1 cm because net power was nearly independent of fin length. In addition, as was the case with the strip fins, the fin thickness is typically constrained to discrete values because of manufacturability considerations. This was not used as an optimization paramter.

8.3.4 Strip Fins

Strip fins offer a well understood means of enhancing convection heat transfer, and they were modeled as enhancement in both the exhaust and coolant sides. Drawings of a finned plate heat exchanger showing cross-section views in both a stream-wise and isometric orientation are shown in Figure 8.6.



(a) (b)

Figure 8.6: (a) Stream-wise and (b) isometric view of cross-section of strip fins in heat exchanger consisting of coolant and exhaust ducts.

Strip fins were modeled using basic finned surface theory [40]. First, the heat transfer coefficient was calculated based on the geometrical and flow properties of the new channels formed by the fins. This was then used to calculate the fin efficiency using

$$\eta_{\text{fin}} = \frac{\tanh(\chi)}{\chi} \tag{8.3.15}$$

where

$$\chi = \beta \text{height} \tag{8.3.16}$$

and

$$\beta = \sqrt{2h_{\text{conv}}/(k_{\text{fin}}\text{thickness}} \tag{8.3.17}$$

Height and thickness are the fin dimensions, β is a dimensionless fin parameter, h_{conv} is the coefficient of convection from Equation 8.2.29, χ is a scaled fin length, and η_{fin}

is the fin efficiency. The fin effectiveness, i.e. the ratio of heat transfer with the fin to heat transfer from an identical surface without the fin, was calculated using [40]:

$$\epsilon_{\mathrm{f}} = 2\eta \frac{\text{height}}{\text{thickness}} \tag{8.3.18}$$

The composite convection coefficient, which accounted for the fin effectiveness, of the base was calculated using

$$h_{\mathrm{comp}} = \frac{h_{\mathrm{unfinned}} \epsilon_{\mathrm{f}} A_{\mathrm{finned}} + h_{\mathrm{unfinned}} A_{\mathrm{unfinned}}}{A_{\mathrm{finned}} + A_{\mathrm{unfinned}}} \tag{8.3.19}$$

where h_{unfinned} is the heat transfer coefficient to all surfaces without consideration of fin effects, A_{finned} is the base area of the finned surfaces, and A_{unfinned} is the area of the channel that is unfinned on both sides.

Pressure drop was modeled using the same correlation as was used for the open channel, with hydraulic diameter and other geometry variables recalculated as appropriate for the channels formed by the fins. Because the strip fin configuration resulted in an abrupt change in flow channel geometry, Equation (8.3.7) was used to account for minor losses at the inlet and exit. For this configuration, the loss coefficients for the inlet and exit were expressed as [80]

$$K_{\mathrm{in}} = \left(K_c + 1 - \left(\frac{A_{\mathrm{flow}}}{A_0} \right)^2 \right) \tag{8.3.20}$$

and

$$K_{\mathrm{out}} = -\left(1 - \left(\frac{A_{\mathrm{flow}}}{A_0} \right)^2 - K_e \right) \tag{8.3.21}$$

where K_c (0.4) and K_e (0.2) are loss coefficients for the inlet and exit, respectively; A_{flow} is the actual flow area in the finned section; and A_0 is the nominal frontal area of the finned section.

For the optimization of strip fins, fin spacing was used as an additional optimization parameter, but fin thickness was not optimized. This was due to the

fact that fin thickness is typically limited to a few discrete values as determined by manufacturing constraints or availability of off-the-shelf parts.

Chapter 9

Model Results and Discussion

9.1 Design Space

As indicated in Section 8.2.5, n-/p-type leg area ratio, leg length, fill fraction, and current (as a surrogate for load resistance) were all used as optimization parameters for maximizing net power output. Optimal n-/p-type leg area ratio remained relatively constant with respect to the other parameters. This is because temperature dependency of the thermoelectric properties exhibited similar trends for both the n- and p-type materials. The latter three parameters, leg length, fill fraction, and current, exhibited some strongly interdependent tendencies in the way they affected power output, and to demonstrate this, surface projections of the 3-dimensional design space are plotted in Figure 9.1. The optimal values for each parameter in the design space are shown in Table 9.1. For this design space, n-/p-type leg area ratio is held constant at 0.7 because there is almost no change in optimal leg area as the other parameters are varied. Convection boundary conditions are 300 K and 680 K with overall heat transfer coefficients of 8 $\frac{\text{kW}}{\text{m}^2\text{K}}$ and 2 $\frac{\text{kW}}{\text{m}^2\text{K}}$ for the coolant and exhaust, respectively. These boundary condition are typical values for coolant and exhaust averaged in the stream-wise direction in the model heat exchangers.

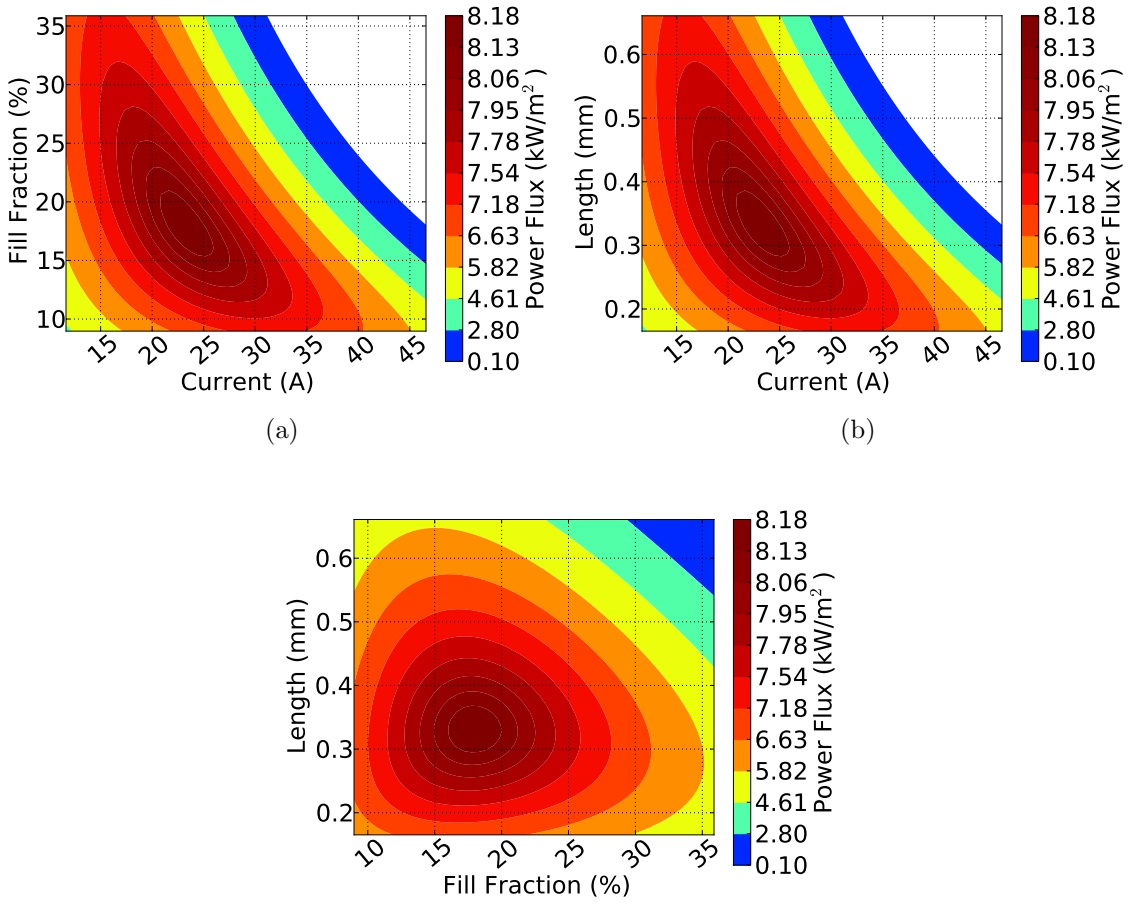


Figure 9.1: 3D Surface plots showing thermoelectric power output v. (a) current and fill fraction, (b) current and length, and (c) fill fraction and length. n-/p-type leg area ratio is held constant at 0.7. Convection boundary conditions are 300 K and 680 K with overall heat transfer coefficients of 8 $\frac{\text{kW}}{\text{m}^2\text{K}}$ and 2 $\frac{\text{kW}}{\text{m}^2\text{K}}$ for the coolant and exhaust, respectively.

Table 9.1: Optimal parameters for design space of standalone TE device, as determined by model. Convection boundary conditions are 300 K and 680 K with overall heat transfer coefficients of 8 $\frac{\text{kW}}{\text{m}^2\text{K}}$ and 2 $\frac{\text{kW}}{\text{m}^2\text{K}}$ for the coolant and exhaust, respectively.

Parameter	Value
fill fraction	17.9 %
leg length	0.330 mm
current	23.3 A
n-/p-type leg area ratio	0.700

These results indicate that changing any of the three variables (fill fraction, leg length, or current) can greatly impact the effect the other two have on performance. This also shows that there is a clearly defined optimal location within the design space. Leg area ratio is included because it does require optimization, but the optimal n-/p-type leg area ratio is insensitive to the other three parameters as well as boundary conditions.

Note that the results presented in Figure 9.1 vary with convection boundary conditions, and as such, the HX system model would produce results that are somewhat shifted in the design space due to a range of convection boundary conditions throughout the stream-wise direction of the heat exchanger. Exploring this relationship is beyond the scope of the present work.

9.2 System Power Output

The system power output results will be presented for each enhancement strategy. After presenting the results, the enhancement strategies will be discussed in Section 9.2.6. For all cases modeled, the TEM package volume was 29.5 L, the length was 50.8 cm, the width was 50.8 cm, the exhaust duct height was 6.35 cm, the coolant duct height was 2.54 cm, and the exhaust flow enthalpy flux was 122 kW for an inlet temperature of 800 K and a mass flow rate of 7.9 kg/min.

9.2.1 Porous Metal Foams

For porous metal foams as heat transfer enhancement, pumping power requirements were on the order of kilowatts or more, reducing the net power output. Hence, porous metal foams were not considered.

To demonstrate the math behind this, sample values for Equations 8.3.1, 8.3.2, 8.3.3, 8.3.4, and 8.3.5 are shown in Table 9.2.

Table 9.2: Sample values for porous media pressure drop model.

D_{pore}	Re_D	ρ	L	F	G	ΔP
1.73 mm	418	447 g/m^3	55 cm	0.203	6.83 $\frac{\text{kg}}{\text{m}^2\text{s}}$	342 kPa

9.2.2 No Enhancement

For the case with no heat transfer enhancement, the results are shown in Table 9.3, and the optimal design space parameters are shown in Table 9.4.

Table 9.3: Results for heat exchanger without heat transfer enhancement.

Parameter	Value
Net Power	221.5 W
$\dot{Q}$	5.953 kW
TE Power	222.5 W
Pumping Power	0.9693 W
System Efficiency	1.81 %
TE Thermal Efficiency	3.73 %
HX Effectiveness	4.88 %

Table 9.4: Optimal design space parameters for heat exchangerwithout heat transfer enhancement.

Parameter	Value
fill fraction	6.31 %
leg length	0.470 mm
current	21.7 A
n-/p-type leg area ratio	0.703

For this case, the system thermal efficiency, given by dividing the power by the inlet enthalpy flow, is a mere 1.81 %. This is due to low TE thermal efficiency and the fact that without heat transfer enhancement, the heat exchanger effectiveness is quite low.

9.2.3 Impinging Jets

For the case with impinging jets as heat transfer enhancement, the results are shown in Table 9.5, and the optimal design space parameters are shown in Table 9.6.

Table 9.5: Results for heat exchanger with strip jets asheat transfer enhancement.

Parameter	Value
Net Power	580 W
$\dot{Q}$	18.1 kW
TE Power	687 W
Pumping Power	108 W

Table 9.6: Optimal design space parameters for heat exchangerwith strip jets as heat transfer enhancement.

Parameter	Value
fill fraction	1.77 %
leg length	0.579 mm
current	17.8 A
n-/p-type leg area ratio	0.702
orifice diameter, D	3.22 mm

9.2.4 Offset-strip Fins

For the case with offset-strip fins as heat transfer enhancement, the results are shown in Table 9.7, and the optimal design space parameters are shown in Table 9.8.

Table 9.7: Results for heat exchanger with offset-strip fins asheat transfer enhancement.

Parameter	Value
Net Power	2.11 kW
$\dot{Q}$	57.2 kW
TE Power	2.33 kW
Pumping Power	219 W

Table 9.8: Optimal design space parameters for heat exchangerwith offset-strip fins as heat transfer enhancement.

Parameter	Value
fill fraction	32.0 %
leg length	2.57 mm
current	4.13 A
n-/p-type leg area ratio	0.700
fin spacing, s	2.12 mm

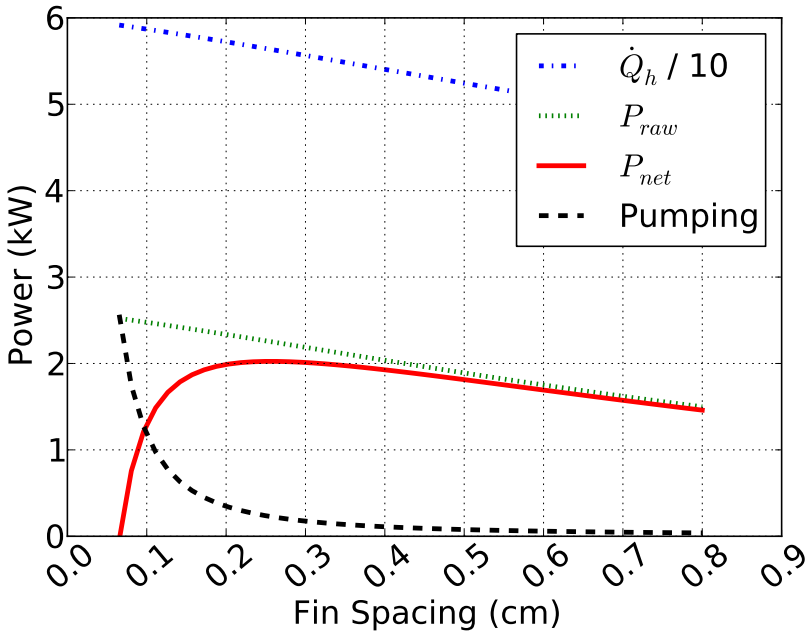


Figure 9.2: Plot of net power, pumping power, raw power, and hot side heat transfer rate v. fin spacing for offset-strip fins.

In Figure 9.2, the pumping power initially decreases sharply with increasing fin spacing due to lower wetted area in the flow path and greater flow area. However, beyond the optimum spacing of 2.12 mm, the loss in heat transfer and TE power becomes more important and the net power decreases.

9.2.5 Strip Fins

For the case with strip fins as heat transfer enhancement, the results are shown in Table 9.9, and the optimal design space parameters are shown in Table 9.10.

Table 9.9: Results for heat exchanger with strip fins asheat transfer enhancement.

Parameter	Value
Net Power	2.25 kW
$\dot{Q}$	58.2 kW
TE Power	2.41 kW
Pumping Power	158 W

Table 9.10: Optimal design space parameters for heat exchangerwith strip fins as heat transfer enhancement.

Parameter	Value
fill fraction	37.5 %
leg length	3.01 mm
current	3.59 A
n-/p-type leg area ratio	0.700
fin spacing	1.83 mm

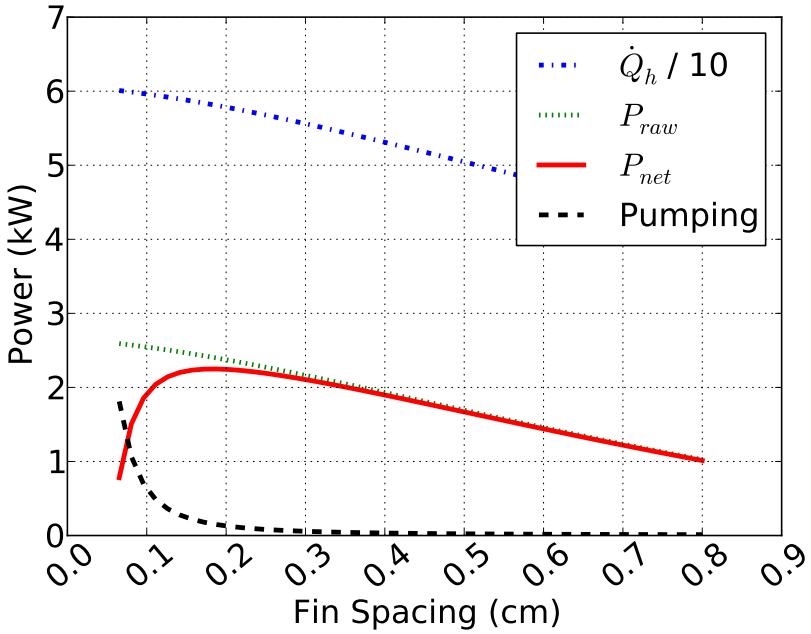


Figure 9.3: Plot of net power, pumping power, raw power, and hot side heat transfer rate v. fin spacing for strip fins.

In Figure 9.3, the pumping power decreases sharply initially due to lower wetted area in the flow path and greater flow area. However, beyond the optimum spacing of 1.81 mm, the loss in heat transfer and TE power becomes more important and the net power decreases. For the strip fins, the optimum spacing occurs at a smaller fin spacing because pressure drop and heat transfer are smaller for strip fins than for offset-strip fins of the same spacing.

9.2.6 Enhancement Discussion

For the conditions considered, the net power results are summarized in Table 9.11 in descending order.

Table 9.11: Net power for all hot-side enhancement configurations.

Enhancement	Net Power
Strip Fins	2.25 kW
Offset-strip Fins	2.11 kW
Impinging Jets	580 W
None	221.5 W
Porous Metal Foams	less than zero

The strip fins and offset-strip fins have nearly the same TE power, but the strip fins have slightly lower pressure drop, resulting in more net power. The two designs are so close in performance that it is possible that offset-strip fins may be better than strip fins for lower flow rates where pressure drop is a smaller penalty. The lower performing enhancement options reduce performance not only in that they result in less heat transfer, but they also result in hot- and cold-side boundary temperatures that are closer together and therefore result in lower efficiency. This temperature dependence of efficiency was shown in Equation 8.1.1.

Chapter 10

First Generation Experiments

10.1 Experimental Apparatus

10.1.1 Engine Test Bed

A Cummins 6.7 L diesel engine was used as the test bed for the heat exchanger experiments. The engine had a maximum rated power output of 350 bhp at 2600 RPM. The maximum speed was 2800 RPM, and a maximum torque of 1020 Nm (750 ft-lbs) was ECU-limited. Brake mean effective pressure at peak torque was 1910 kPa. Bore and stroke were 107 mm and 124 mm, respectively. It was mated to a SuperFlow water-absorption dynamometer. The dynamometer had maximum nominal power absorption of 1000 bhp, but the maximum torque was limited to 680 Nm and was speed dependent, not developing full torque until 2200 RPM. Thus the maximum engine load in this study was limited by the dynamometer up to relatively high engine speed ($\sim$2000 rpm). For engine speed greater than $\sim$2000 rpm, the engine torque decreased rapidly with engine speed, and the maximum torque was limited to less than 680 Nm by the engine.

For information on how the Cummins engine is operated, see Appendix D.

10.1.2 Heat Exchanger

The first generation heat exchanger that was designed and tested for this study was scaled down in size from what would be needed for an actual vehicle application, and its shape was designed to be compatible with flat wafer-like TE devices, at least

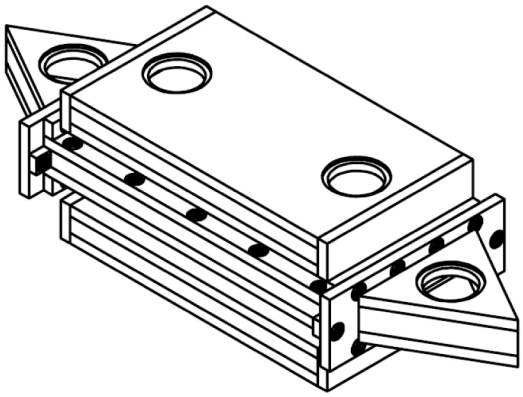
initially, as part of TE device-level research being conducted in unison with this work. An important practical consideration for a production thermoelectric device is the ability of the heat exchanger to resist thermal cycling that can lead to fatigue failure of the individual TE devices due to differential expansion between the semiconductor devices and the metal heat exchanger interface these devices are bonded to. This issue was not considered in the present study. The purpose of the experimental work was to provide validation of the heat exchanger model which will be used to investigate and optimize new heat exchanger designs. A scaled-down version was constructed for reasons of cost and experimental convenience. The dimensions of the heat exchanger are shown in Table 10.1.

Table 10.1: Heat exchanger dimensions.

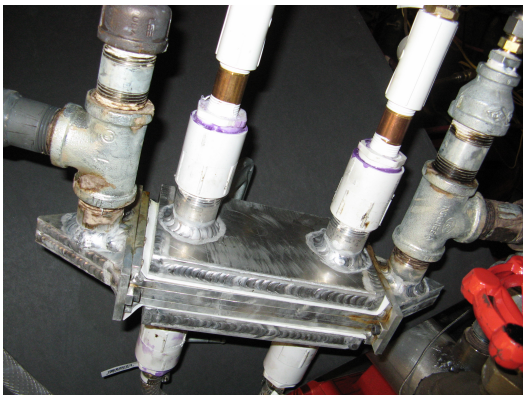
Dimension	
Displaced volume	1.1 L
Exhaust channel height	1 cm
Coolant channel height	0.5 cm
HX width	9.5 cm
HX length	19.5 cm
Alumina paper thickness	1 mm

The flat plate aluminum heat exchanger was fabricated from welded quarter inch aluminum plate. The heat exchanger consisted of a core hot-section through which a portion of the engine exhaust flowed, sandwiched between the two water-cooled cold-sides. A sheet of alumina paper was placed between the hot-exhaust channel and each of the two coolant- channels. The dimensions of the rectangular cross-sectioned heat exchanger flow channels are shown in Table 1. A SolidWorks isometric drawing of the heat exchanger is shown in Figure 10.1a and photos of the heat exchanger are shown in Figures 10.1b and 10.1c. The 1 mm thick alumina paper was chosen as a surrogate for the TE devices, having an estimated equivalent

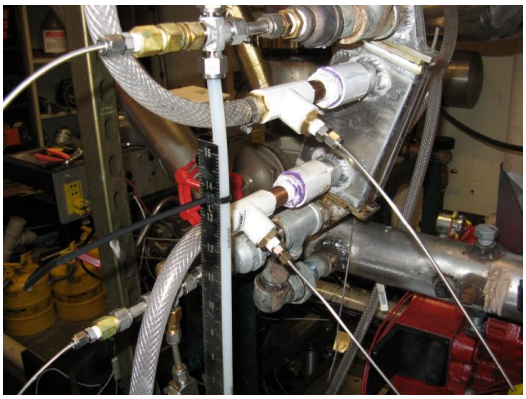
thermal resistance as the n-type ($MgSi_2$) and p-type ($MnSi_{l.75}$) TE materials under development as part of this project. Actual silicide TE devices were not available at the time.



(a)



(b)



(c)

Figure 10.1: (a) Isometric view of heat exchanger. (b) Photograph of heat exchanger as installed. (c) Thermocouples sheathed in 1/8 in diameter tubing are shown inserted into the entrance and exit pipes on both the hot and cold sides. The hot-side thermocouples extend to the centerlines of the triangular heat exchanger end caps.

The heat exchanger rig was outfitted with pressure taps and thermocouples both upstream and downstream and in both the coolant and exhaust ducts. A schematic of the heat exchanger rig is shown in Figure 10.2.

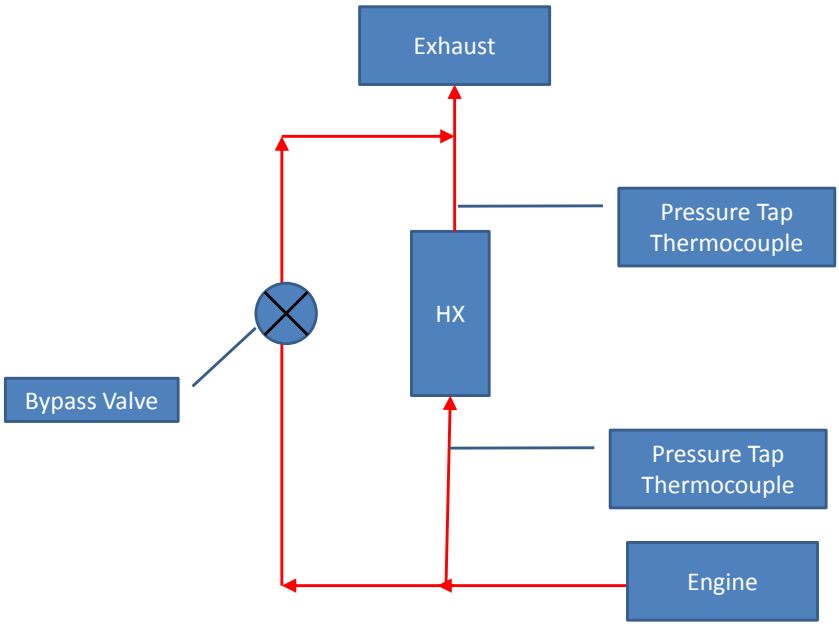


Figure 10.2: Schematic of the heat exchanger rig.

10.2 Experimental Methods

10.2.1 Heat Exchanger Operation and Measurements

For the first generation heat exchanger, only a portion of the total Diesel exhaust was flowed through the heat exchanger in order to avoid excessive back pressure due to the small size. The flow was regulated by partially closing or opening the bypass gate valve shown in Figure 10.2.

The volume flow rate was measured directly using a bell jar and a stop watch. The temperature of the gas in the bell jar was measured using a thermocouple to determine the density of the measured gas relative to that of the hot gas flowing in the heat exchanger. A photo of the bell jar measurement device is shown in Figure

10.3.

Figure 10.3: Photo of bell jar volume flow rate measurement apparatus.

The pressure drop was calibrated to the volume flow rate determined by the bell jar experiments, eliminating the need for direct volume measurement in subsequent experiments.

Building water at approximately 300 K and total flow rate of 15 Lpm was used for the coolant through the cold-sides of the heat exchanger. The exhaust gas flow rate through the hot-side of the heat exchanger was varied between approximately 20 L/s and 60 L/s as an independent variable.

The temperature and flow measurements were taken at steady-state conditions, thus, the thermal inertia of the system was not considered to be a significant influence. The heat exchanger system was considered in steady-state when the hot-side inlet and outlet temperatures were changing by no more than 1 °C per minute. It typically took approximately 10 minutes to achieve this rate between changes in engine operating conditions or hot-side exhaust gas flow rate.

10.3 Results and Model Validation

The result of the exhaust side pressure drop (measured using the two pressure taps shown in Figure 10.2) versus flow rate calibration is shown in Figure 10.4.

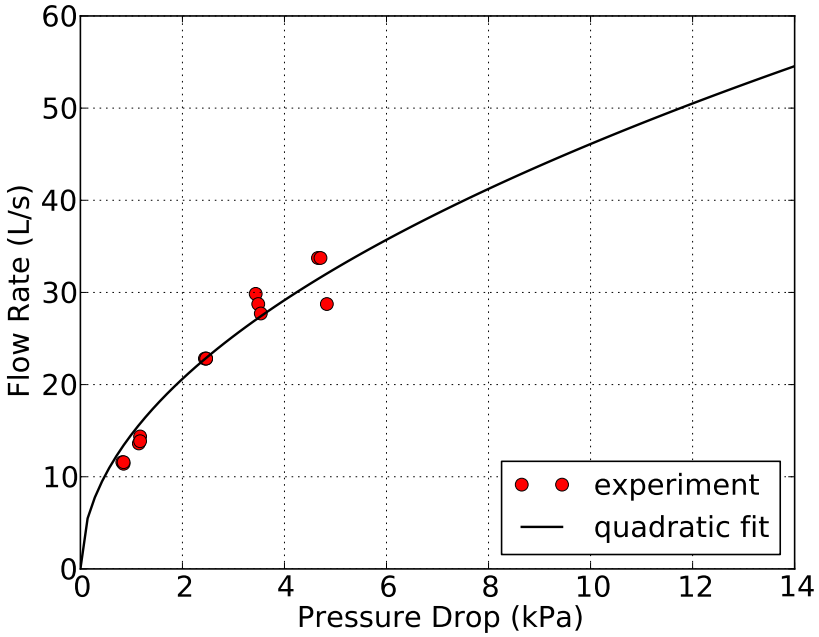


Figure 10.4: Pressure drop versus flow rate calibration results.

The quadratic fit obtained from this experiment was then used to determine flow rate as a function of pressure drop for all subsequent experiments on this heat exchanger.

The goal of the experimental work was to measure heat transfer rates from the hot engine exhaust gases to the coolant flow over a range of inlet temperatures considered typical for on-road engine operation, and to determine the effectiveness of the heat exchanger for a range of scaled exhaust flow rates. All of the experimental results were gathered at either of three torque values all at 1400 rpm and without EGR. Hot- side inlet temperature as a function of flow rate is shown in Figure 10.5.

Variation in engine load was the means used to vary the hot-side inlet temperature, and the gate valve shown in Figure 10.2 was the means used to vary the flow rate.

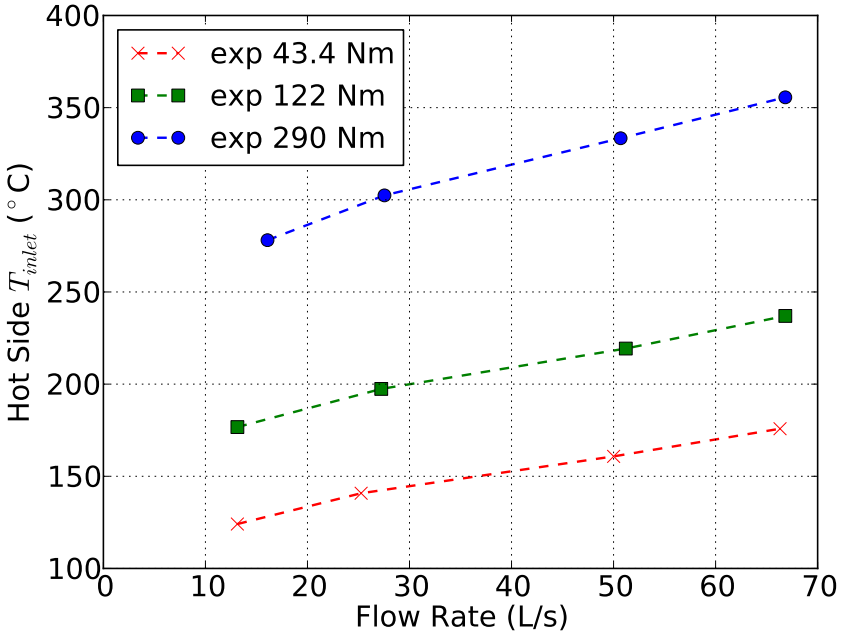


Figure 10.5: Exhaust inlet temperature versus flow rate for 1400 rpm. Torque values shown in legend.

The highest torque load examined, 290 Nm, was a lower mid-load value at 1400 rpm. Figure 10.5 shows the extent that the exhaust temperature at the inlet of the heat exchanger increases with engine load. The inlet temperature increases with increasing flow rate because of reduced heat transfer losses to the entrance piping associated with the increasing heat capacity flow rate and reduced residence time. Figure 10.6 shows the temperature drop across the hot side of the exchanger as a function of flow rate for the three torque values, all at 1400 rpm.

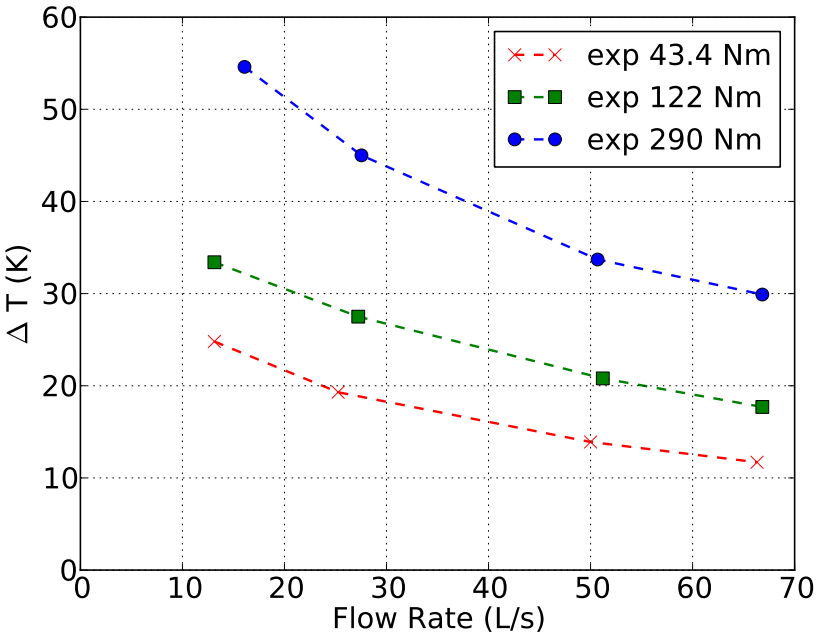


Figure 10.6: Parameterized plot of experimental and model temperature drop in exhaust gas vs. flow rate for 1400 rpm and engine torque values shown in the figure.

The trends are consistent with expectations that higher inlet temperatures, and therefore greater hot-side to cold-side temperature differences, lead to more heat transfer and thus greater temperature drops across the heat exchanger. The temperature drops decrease with increasing flow rate due to the decrease in fluid residence time, and therefore shorter time for heat transfer to occur.

For each case, the heat transfer rate was calculated from a 1st law energy balance across the heat exchanger as the product of the mass flow rate and the change in enthalpy of the exhaust gas across the hot-side. The heat transfer rate versus flow rate from the experiment is compared with the results of the model in Figure 10.7.

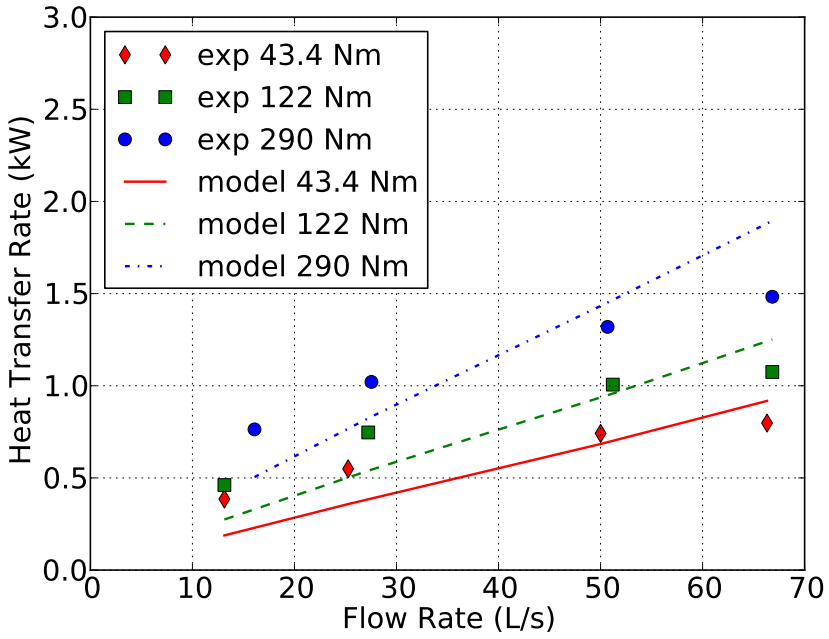


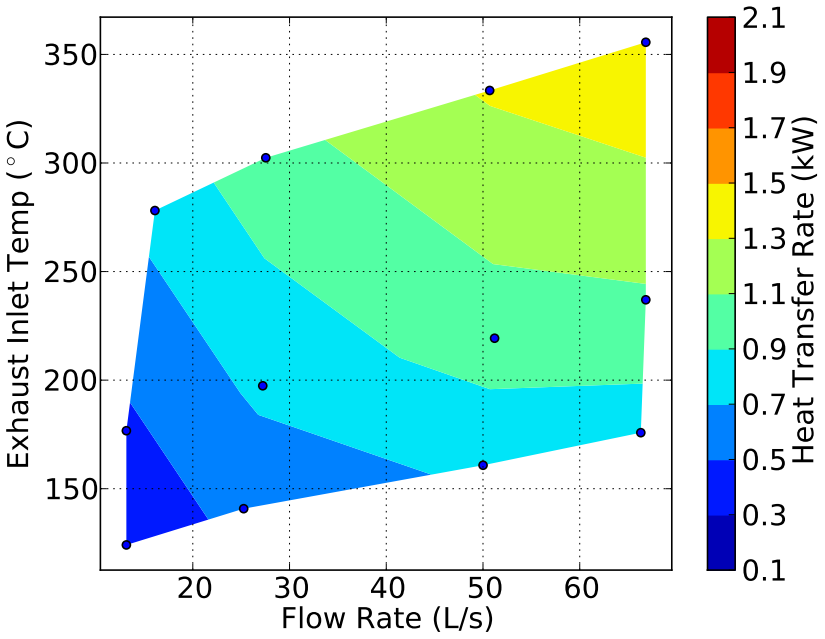
Figure 10.7: Parameterized plot of experimental and model heat transfer from exhaust gas v. flow rate for varied engine torque (1400 rpm).

Heat transfer rate is seen to increase, both with increasing flow rate and engine load due to respective increases in thermal capacity and temperature. At lower flow rates the model under-predicted the experimental heat transfer, and at higher flow rates, the model over-predicted the experimental heat transfer. It is not certain why the experimental results deviate from the model as they do. The thermocouples measuring the exhaust gas temperature were placed upstream and downstream of the hot-side exhaust flow channel, in the pipes that delivered the flow to and from the heat exchanger. One factor which contributed to high measured heat transfer rates at lower flow rates was heat transfer losses in the uninsulated triangular manifold at the entrance to the rectangular cross-section exhaust channel. The triangular manifold at the inlet of the heat exchanger created an impingement surface that results in a locally high heat transfer coefficient. At lower flow rates the flow residence time is longer,

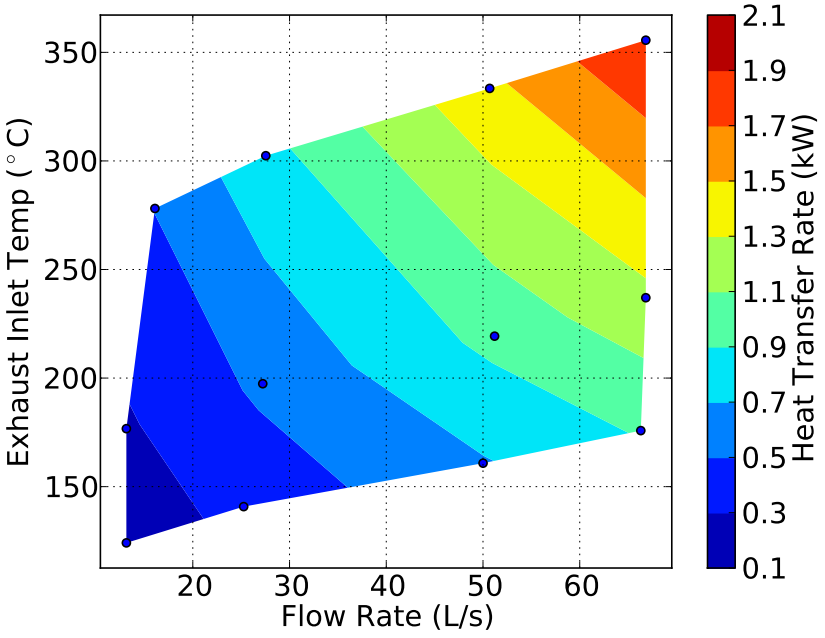
allowing more time for heat transfer in this section, possibly leading to proportionately larger heat transfer upstream of the rectangular section. Heat transfer through the side-walls of the heat exchanger also contributed. At higher flow rates the measured heat transfer rates were below the model predictions. Contact resistance between the hot-side and cold-side was neglected. This would have contributed to the measured heat transfer rates being low, but it is not known by how much since the contact resistance was unknown.

A conceptual limitation in viewing the results of Figure 10.7 is that plotting heat transfer rate as a function of torque does not show how the inlet temperature is varying. While Figure 10.5 gives the inlet temperature associated with each data point in Figure 10.7, a more integrated view of the results is provided by a two-dimensional contour plot as shown in Figure 10.8.

Figure 10.8 shows the combined results of Figures 10.5 and 10.7, where exhaust inlet temperature is used as an independent variable rather than engine torque. Figure 10.8 shows that the range of heat transfer rates predicted by the model is greater than the heat transfer rate calculated from the experimental data. The minimum and maximum heat transfer rates calculated from the model were 188 W and 1.89 kW, respectively. The calculated minimum and maximum heat transfer rates from the experimental results were 385 W and 1.45 kW. The heat transfer rate predicted by the model has a stronger dependence on both flow rate and inlet temperature.



(a) Experiment



(b) Model

Figure 10.8: Two-dimensional surface plot of (a) experimental and (b) model heat transfer rate from exhaust gas v. flow rate and exhaust temperature at inlet of heat exchanger.

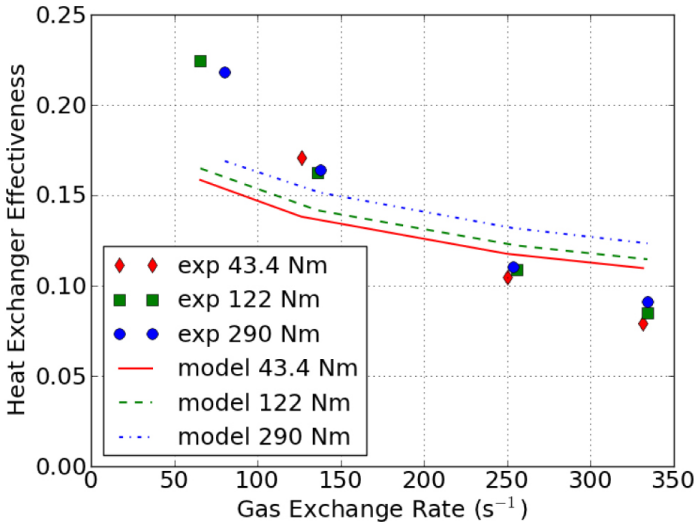
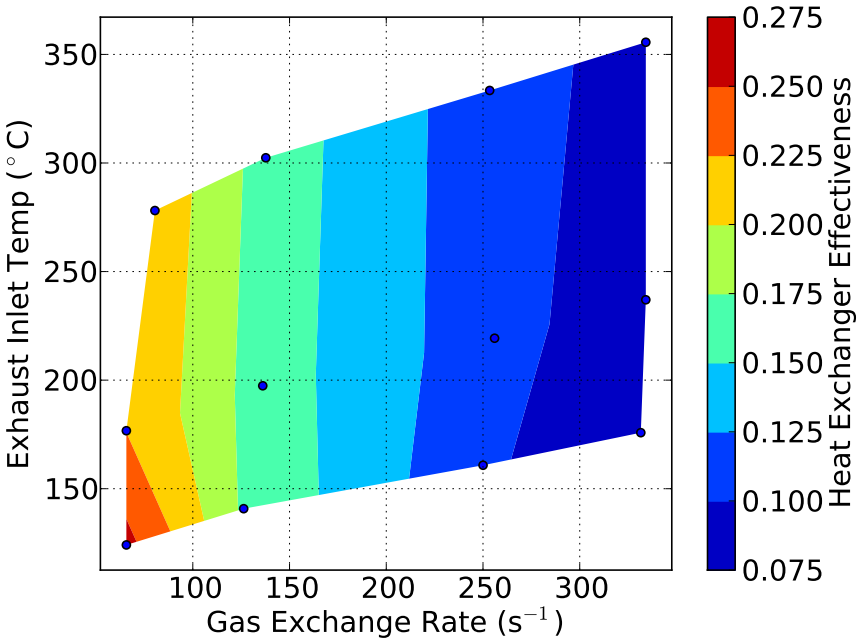


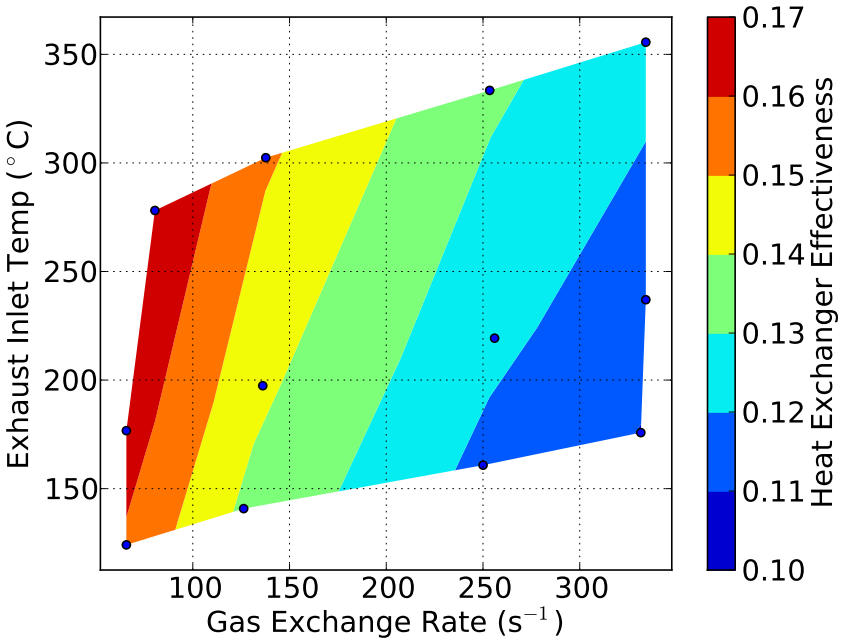
Figure 10.9: Parameterized plot of experimental and model heat exchanger effectiveness v. gas exchange rate for 1400 rpm and varied engine torque. Torque values shown in legend.

Figure 10.9 shows that the heat exchanger effectiveness of the model has less dependence on flow and more dependence on engine load compared to the heat exchanger effectiveness of the experimental results. This result is consistent with the hypothesis that the impinging flow created by the inlet pipe caused high heat transfer with the surroundings and not just to the coolant.

By non-dimensionalizing heat transfer rate, the heat transfer characteristics can be more generally compared with other heat exchangers and engines. To facilitate this, the standard definition heat exchanger effectiveness was calculated by normalizing the heat transfer rate by the hot-side mass flow rate multiplied by the enthalpy difference of the exhaust gas between the hot-side inlet temperature and the inlet temperature of the coolant. The exhaust flow rate through the hot-side was divided by the hot-side volume to yield gas exchange rate. Results from this scaling are plotted in Figure 10.10.



(a) Experiment



(b) Model

Figure 10.10: Two-dimensional surface plot of (a) experimental and (b) model heat exchanger effectiveness v. gas exchange rate and exhaust temperature at inlet of heat exchanger.

Figure 10.10 shows that the heat exchanger effectiveness predicted by the model has a greater dependence on both flow and inlet temperature, while the experimental results show almost no dependence on inlet temperature. In Figure 10.9, the experimental heat exchanger effectiveness shows a small dependence on engine load, consistent with Figure 10.10a.

The experimental flow friction factor for the heat exchanger was determined using the manometer calibration and Equation 8.2.31. A plot of friction factor versus Reynolds number for both the experiment and the model is shown in Figure 10.11.

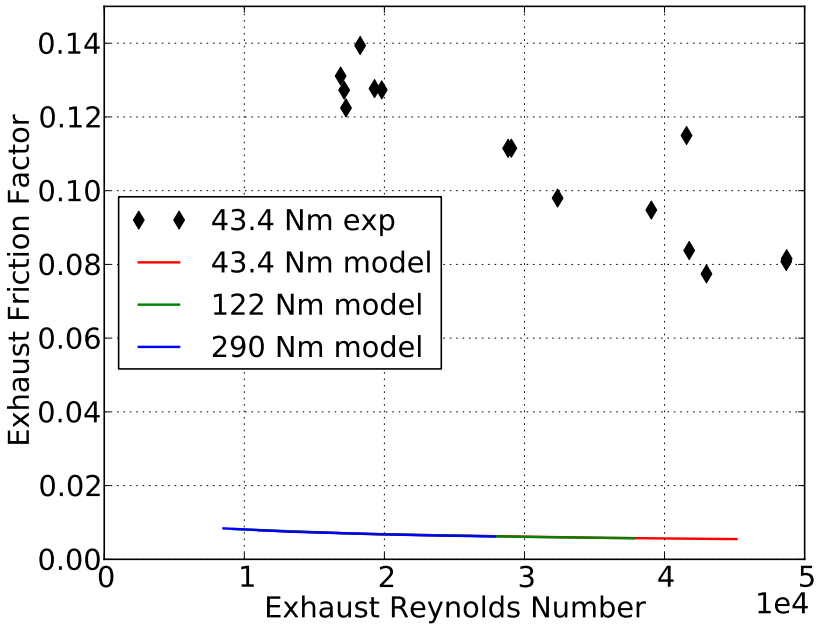


Figure 10.11: Friction factor v. Reynolds number for both model and experimental results for 1400 rpm. Torque values are shown in the legend.

Figure 10.11 shows that the friction factor predicted by the model is approximately 1/15th of the experimental friction factor. This is likely due to minor and major losses that occur in the flow path between the pressure taps and the rectangu-

lar channel section handled by the model. The model does not attempt to account for the losses in the pipes that carry the flow to and from the heat exchanger which have a cross-sectional flow area that is approximately half that of the heat exchanger exhaust channel. For the heat exchanger that will be discussed in Section 11, there will be greater effort to ensure that the inlet and outlet flow do not cause a significant amount of pressure drop compared to that of the heat exchanger duct.

Chapter 11

Second Generation Experiments

11.1 Experimental Apparatus

The second generation heat exchanger experiment utilized the same engine and dynomometer rig. There were, however, substantial changes and improvements in the heat exchanger that will be discussed subsequently.

11.1.1 Heat Exchanger

The second generation heat exchanger was much larger than the first generation heat exchanger in order to make use of the entire flow produce by the Cummins 6.7 L diesel engine. This heat exchanger consisted of a flat plate exhaust duct sandwiched between two coolant ducts with two different TE surrogate materials between the ducts. The surrogate materials were used because the materials science component of this project has not yet produced TE devices for this purpose.

A photo of the second generation heat exchanger are shown in Figure 11.1, an engineering drawing of the second generation heat exchanger is shown in Figure 11.2, and a schematic of the heat exchanger showing the flow path is shown in Figure 11.3.

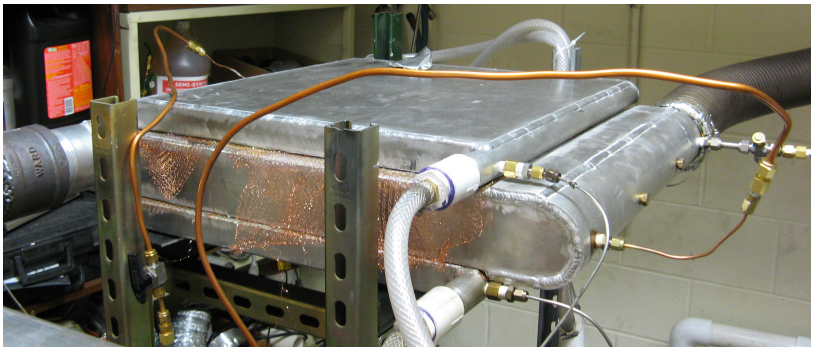


Figure 11.1: Photo of the second generation heat exchanger, as installed inline with the Cummins exhaust system. The heat exchanger is downstream of the turbocharger turbine.

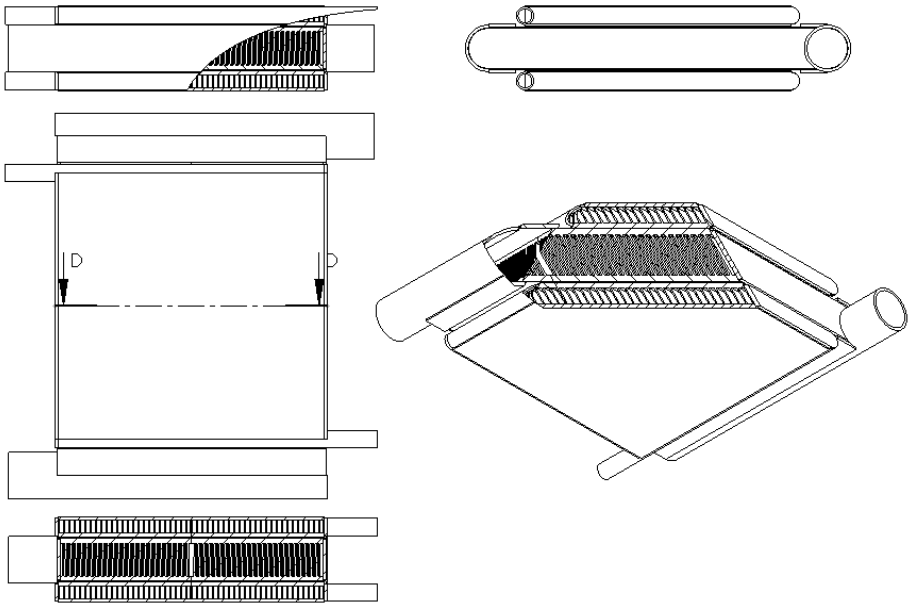


Figure 11.2: Engineering drawing of the 2nd generation heat exchanger.

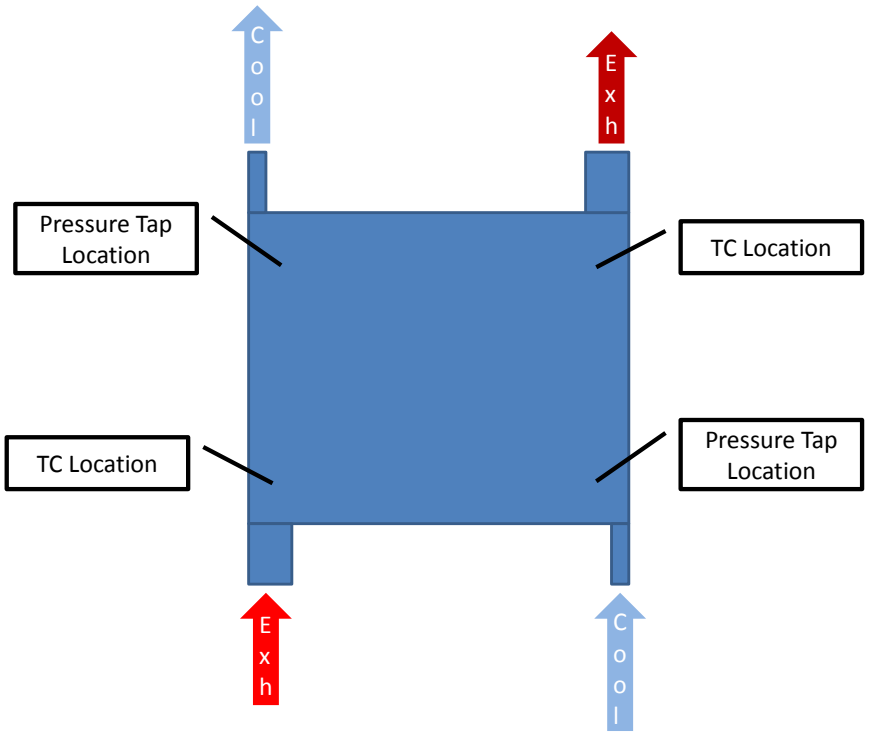


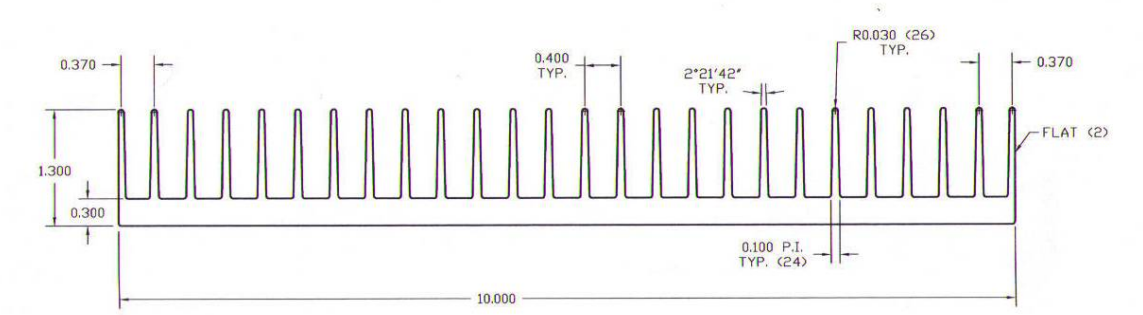
Figure 11.3: Top view schematic of the second generation heat exchanger. Arrows indicate flow. The pressure tap consists of a 1/8 in copper tube with the opening facing the vertical direction perpendicular to the flow (out of the page as viewed by the reader).

The exhaust flow entered through a 3" pipe at the bottom left of the figure. As it flowed along the pipe, it was forced to flow to the right through the center of the channel where all of the finned surfaces are. After leaving the fins, the exhaust was collected by the downstream pipe, where it began flowing in an upward direction with respect to the orientation of the figure. By having the inlet and exit arranged in this diagonally opposed fashion, the pressure drop through the fins was uniform in the vertical direction with reference to the figure orientation. This encouraged uniform flow through the center section containing the fins. The exhaust duct had finned surfaces on both the top and bottom sides, and each set of fins was in contact

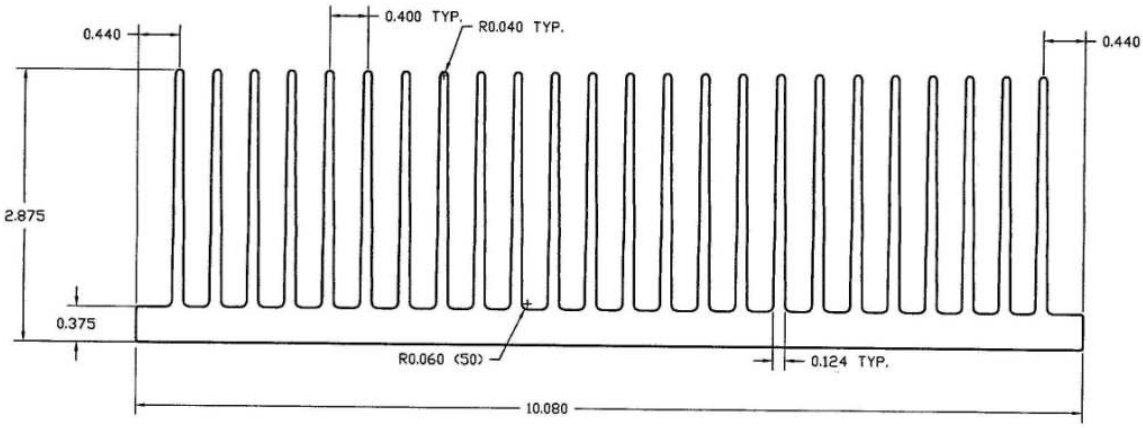
with the opposite side.

There were two coolant ducts with flow in the opposite direction of the exhaust. One was underneath (behind in the figure) the exhaust duct, and the other was above (in front in the figure) the exhaust duct. The coolant ducts had fins that extended all the way across the channel on only one side - the side on which heat transfer was desired.

Extruded finned aluminum plates were used as sides of the heat exchanger that had fins. Profile views of the fins for both the coolant and exhaust ducts are shown in Figure 11.4.



(a) 10” - coolant



(b) 10.08” - exhaust

Figure 11.4: Manufacturing drawing of prfile view of (a) 10 inch wide finned aluminum extrusion used in the coolant duct and (b) 10.08 inch wide finned aluminum extrusion using in the exhaust duct [81].

The heat exchanger consisted of two 10 inch profiles aligned in the span-wise direction with a 22 inch length for the enhancement on each cold side and two sets of two 10.08 inch profiles aligned in the span-wise direction and stacked to face each other with the fins offset such that each fin was inserted in the corresponding slot of the opposing plate. This resulted in an exhaust duct height of 2.5 inches, or the same height as the fins. Detailed engineering drawings of this heat exchanger are shown in Appendix C.

Either copper mesh or gypsum board was used as a surrogate material to emulate the thermal resistance of TE devices, which were not ready for use as of this writing. These materials were selected to encompass the entire range of possible TE device thermal resistance, which can vary widely with leg height and fill fraction as well as material properties. In addition, the large range of thermal conductivity between these two materials provides a stronger validation of the model.

The assumed thermal conductivity for gypsum was chosen to be 17 W/m-K based on Incropera and DeWitt [66]. For 1/4" gypsum board with this thermal conductivity, the thermal resistance was 58.8 m^2-K/kW. This thermal resistance did not account for contact resistance. The thermal conductivity for the copper mesh was estimated to be between 34.1 mW/m-K and 20.0 W/m-K using [82]

$$k_{\text{lower}} = \left(\frac{\phi_{\text{Cu}}}{k_{\text{air}}} + \frac{1 - \phi_{\text{Cu}}}{k_{\text{Cu}}} \right)^{-1} \tag{11.1.1}$$

for the lower and

$$k_{\text{upper}} = \phi_{\text{Cu}} k_{\text{air}} + (1 - \phi_{\text{Cu}})\, k_{\text{Cu}} \tag{11.1.2}$$

for the upper limit. ϕ_{Cu} is the porosity of the copper mesh, k_{air} is the thermal conductivity of air at the average temperature of the exhaust and coolant flows, and k_{Cu} is the thermal conductivity of bulk copper (400 W/m-K). The porosity of the copper mesh was calculated by measuring the thickness of a 6" by 6" square of copper to determine the volume. The mesh was weighed, and the weight was compared to the nominal weight of an equal sized piece of solid copper to determine the porosity. The thickness was measured to be 500 μm, and the resulting porosity was 95 %. The thermal resistance was between 25.4×10^{-3} m^2-K/kW and 14.9 m^2-K/kW based on the measured thickness and the thermal conductivity as determined by Equations 11.1.1 and 11.1.2. The range of three orders of magnitude in these estimates is a result of large uncertainty in the various properties of the mesh. These properties consist

of the mesh number, the contact conditions between individual wires and layers of the mesh, the wire diameter, the wire shape, and the compression of the mesh [82]. An estimated range of thermal resistance this large is obviously of little use, so the thermal resistance had to be measured directly.

The thermal resistance of the copper was determined using either the heat exchanger model model or additional experimental results. The first method to do this was to use the thermal resistance of the copper mesh as a fit parameter for the model to match the experimental results. A numerical minimization algorithm was used to determine the value of the thermal resistance (including contact resistance) that best fit the experimental data over the entire temperature range. The other method was to directly measure the thermal resistance using a device that will be described in Section 11.1.2.

11.1.2 Thermal Resistance Measurement Apparatus

To directly measure the thermal resistance of the copper mesh, we constructed a heat flux measurement device shown in Figure 11.5.

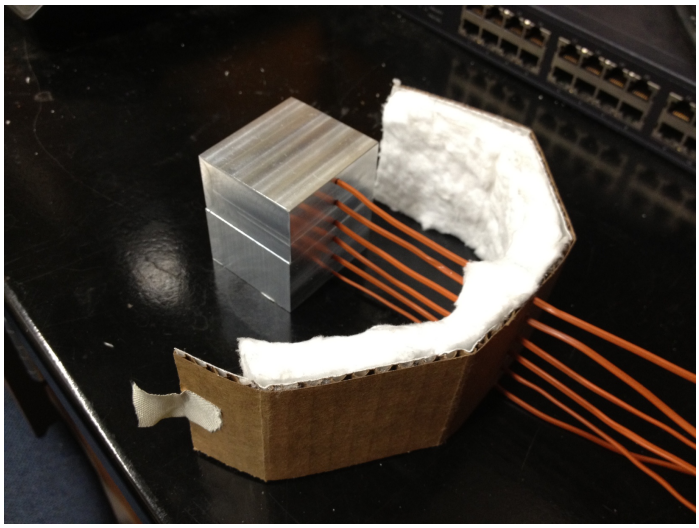


Figure 11.5: Photo of the thermal resistance measurement device.

The device consisted of a heater (not shown), a hot side aluminum block, a cold side aluminum block, and a coolant channel (not shown). Each aluminum block was fitted with three thermocouples spaced evenly in the direction of the temperature gradient. The copper mesh was placed between the two aluminum blocks. The entire assembly was wrapped in alumina insulation that was held in place with a cardboard sleeve and tape.

11.2 Experimental Methods

The engine operation procedures for this second generation heat exchanger were the same as for the previous generation. See Section 10.1.1.

11.2.1 Heat Exchanger Operation and Measurements

Building water at approximately 300 K and flow rate of 26.5 Lpm was used for the coolant through the cold-sides of the heat exchanger. The exhaust gas flow rate through the hot-side of the heat exchanger was varied between approximately 4 and 8 kg/min. The mass flow was taken from the Cummins Calterm III engine calibration software. Volume flow rate was calculated based on temperature measurements.

The temperature and flow measurements were again taken at steady-state conditions. The heat exchanger system was considered in steady-state when the hot-side inlet and outlet temperatures were changing by no more than 1 °C per minute. It typically took approximately 30 minutes to achieve this rate between changes in engine operating conditions or hot-side exhaust gas flow rate. The time constant for this experiment was much larger than the previous generation due to the fact that there was much more mass in the heat exchanger to absorb heat.

11.3 Results and Model Validation

For both gypsum and copper mesh, the results for the heat transfer and pressure drop models are presented separately. For the gypsum board, the assumed thermal resistanced based on the literature value of thermal conductivity was used for the first set of results. For the copper mesh, the model used the thermal resistance obtained by direct measurement using the device described in Section 11.1.2. For both materials, in the second set of results, the model used the thermal resistance of the TE surrogate material (including contact resistances) as determined by using the thermal resistance as a fit parameter to fit the experimental heat transfer data. A single value of thermal resistance was fitted for the entire temperature range.

The calculations for the error bars in the figures in this section are explained in Appendix B. For data points with no error bars, the error was sufficiently small to be obscured by the data points. Heat transfer results are presented as a function of net enthalpy flow. Net enthalpy flow is the total amount of heat transfer that would occur if the heat exchanger had an effectiveness of 100%, given by:

$$\dot{H}_{\text{net}} = \dot{m}_{\text{exh}} h_{\text{net}} = \dot{m}_{\text{exh}} \bar{c_{\text{p}}} \left(T_{\text{exh, in}} - T_{\text{cool, in}} \right) \tag{11.3.1}$$

where $\dot{m}_{\text{exh}}$ is the mass flow rate of the exhaust, h_{net} is the net specific enthalpy, $\bar{c_{\text{p}}}$ is the stream-wise averaged specific heat of the exhaust, and the temperatures have been defined previously.

11.3.1 Gypsum TE Surrogate Material

For the gypsum TE surrogate material with a thermal conduction resistance of 58.8 m^2-K/kW, heat transfer versus net hot-side inlet enthalpy flow rate is shown in Figure 11.6, and dimensionless pressure drop is shown in Figure 11.7.

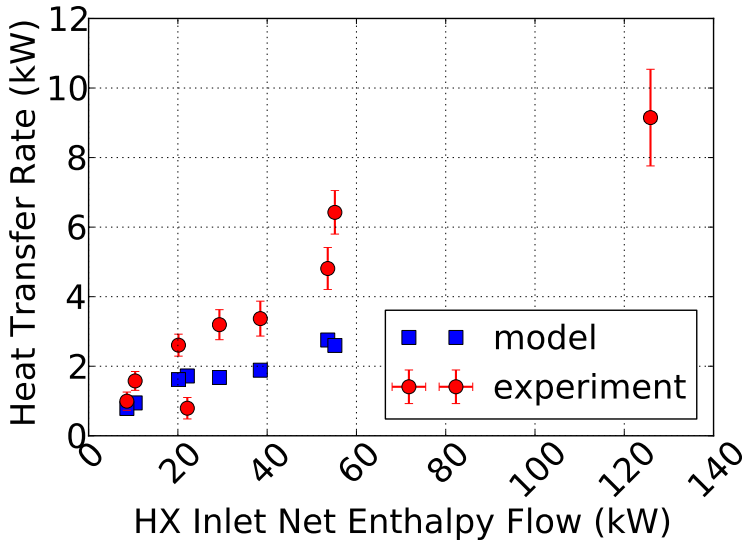


Figure 11.6: Plot of model and experimental heat transfer v. net hot-side inlet enthalpy flow rate for gypsum board as TE surrogate material.

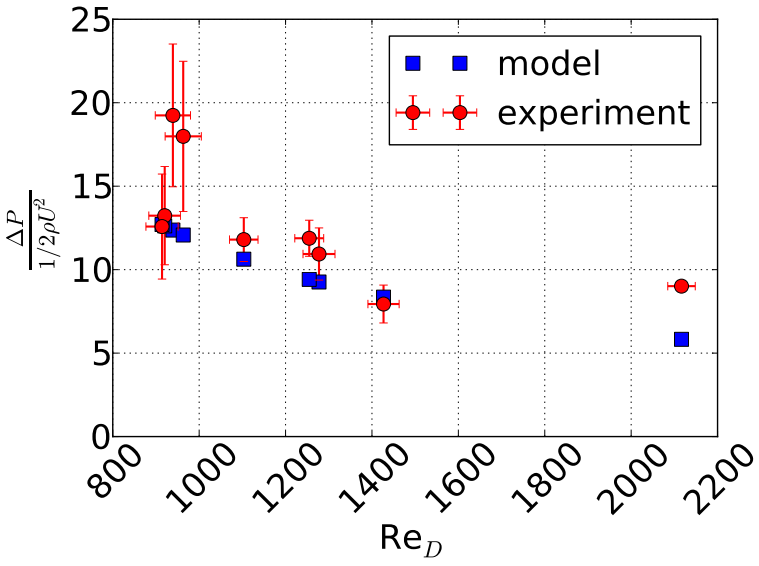


Figure 11.7: Plot of dimensionless pressure drop $\left(\frac{\Delta P}{1/2\rho U^2}\right)$ v. Re_D for model and experimental results with gypsum board as TE surrogate material.

The correlation coefficients between the model and experimental data were

0.891 and 0.737 for Figures 11.6 and 11.7, respectively. Heat transfer and pressure drop were both under-predicted by the model, though the pressure drop data had substantial scatter. To determine if the heat transfer prediction error was caused primarily by the convection model or the assumed thermal resistance of the gypsum board, the thermal resistance of the gypsum was used as a fit parameter in a least squares fit of the experimental data. The thermal resistance determined by this procedure was 17.1 m^2-K/kW. The resulting heat transfer and pressure drop plots are shown in Figures 11.8 and 11.9, respectively.

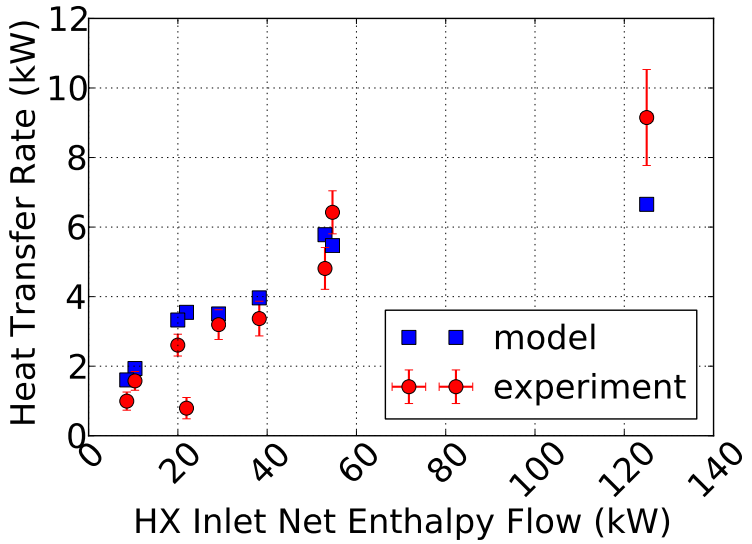


Figure 11.8: Plots of model and experimental heat transfer v. net hot-side inlet enthalpy flow rate for gypsum mesh with fitted thermal resistance as TE surrogate material.

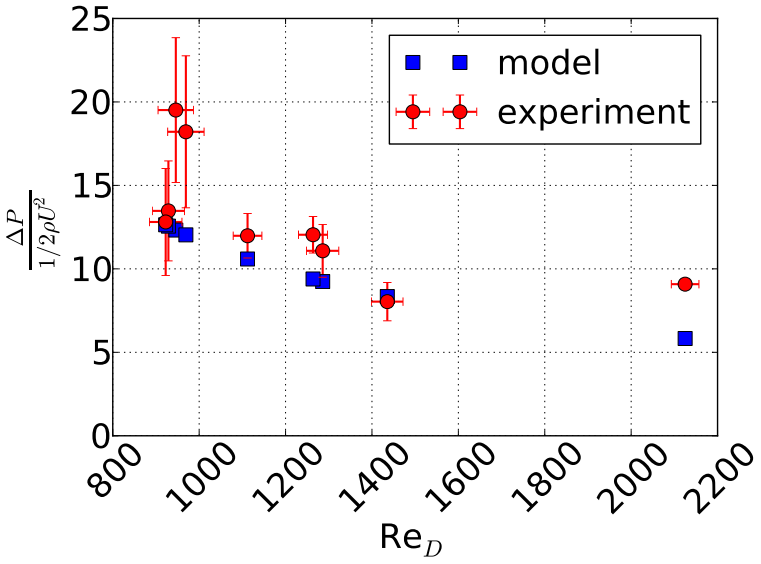


Figure 11.9: Plot of dimensionless pressure drop $\left(\frac{\Delta P}{1/2\rho U^2}\right)$ v. Re_D for model and experiment for gypsum mesh with fitted thermal resistance as TE surrogate material.

The correlation coefficients between the model and experimental data were 0.898 and 0.744 for Figures 11.8 and 11.9, respectively. By using the thermal resistance of the gypsum board as a fit parameter, the heat transfer model was able to fit the experimental data much more effectively. This indicates that the prediction error in Figure 11.6 was due mostly to an incorrect assumption about the thermal resistance the gypsum board. The thermal resistance of the gypsum board determined by the least squares fit was less than the assumed thermal resistance by a factor of 3.4. Because the assumed thermal resistance was higher than the calculated thermal resistance, and the assumed thermal resistance did not include thermal contact resistance, it can be concluded that the thermal conductivity of the gypsum board was higher (i.e. the thermal resistance was lower) than the value presented in Incropera and DeWitt [66]. Had the calculated thermal resistance been higher than the literature value, it would have been possible that the thermal contact resistance of the gypsum

board in the experiment accounted for all of this difference, but this was not the case.

The pressure drop model and experimental results exhibited very little change, and this is expected because the heat exchanger effectiveness was relatively low (less than 10%) both with and without the fitted thermal resistance. This meant that no significant change in volume flow rate occurred, and therefore there was no significant change in pressure drop.

11.3.2 Copper Mesh TE Surrogate Material

The thermal resistance of the Cu mesh measured by the thermal resistance device, including contact resistance, was 4.2 m^2-K/kW. This corresponds to an effective mesh thermal conductivity of 21 W/m-K, and the measured thermal resistance was within the lower (25.4×10^{-3} m^2-K/kW) and upper (14.9 m^2-K/kW) bounds of the thermal resistance predicted by Equations 11.1.1 and 11.1.2. The heat transfer results for this thermal resistance are shown in Figure 11.10, and the dimensionless pressure drop results are shown in Figure 11.11.

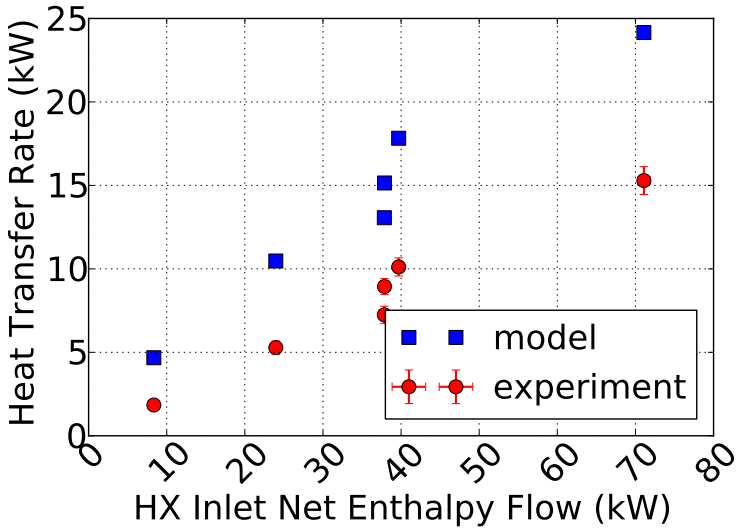


Figure 11.10: Plots of model and experimental heat transfer v. net hot-side inlet enthalpy flow rate for copper mesh with measured thermal resistance as TE surrogate material.

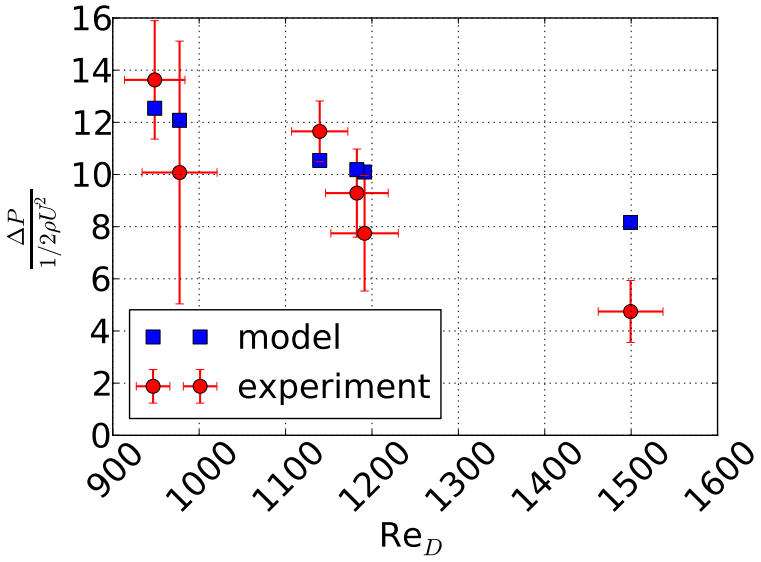


Figure 11.11: Plot of dimensionless pressure drop $\left(\frac{\Delta P}{1/2\rho U^2}\right)$ v. Re_D for model and experiment for copper mesh with measured thermal resistance as TE surrogate material.

The correlation coefficients between the model and experimental data were 0.997 and 0.878 for Figures 11.10 and 11.11, respectively. The heat transfer model consistently over-predicted heat transfer compared to the experimental heat transfer results. This thermal resistance of the copper mesh was also determined by using it as a least squares fit parameter. For the optimal fit of the heat transfer model with the experimental results for the copper mesh, the thermal resistance of was 8.21 m^2-K/kW. With this value of thermal resistance the heat transfer results are shown in Figure 11.12, and the dimensionless pressure drop results are shown in Figure 11.13.

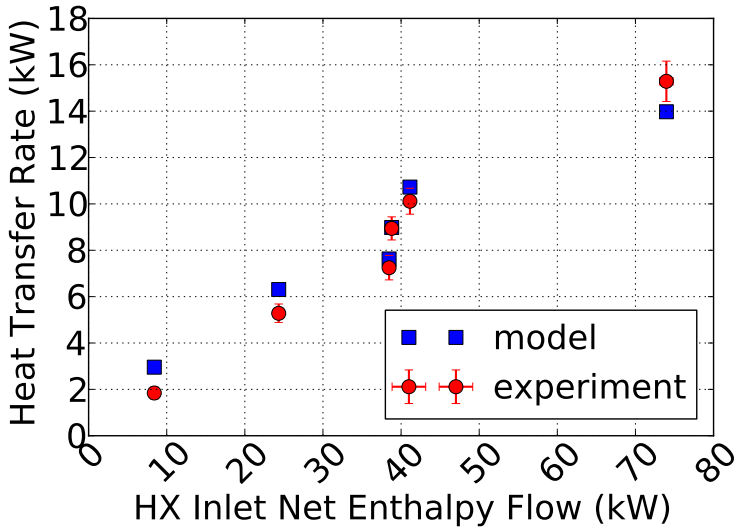


Figure 11.12: Plots of model and experimental heat transfer v. net hot-side inlet enthalpy flow rate for copper mesh with fitted thermal resistance as TE surrogate material.

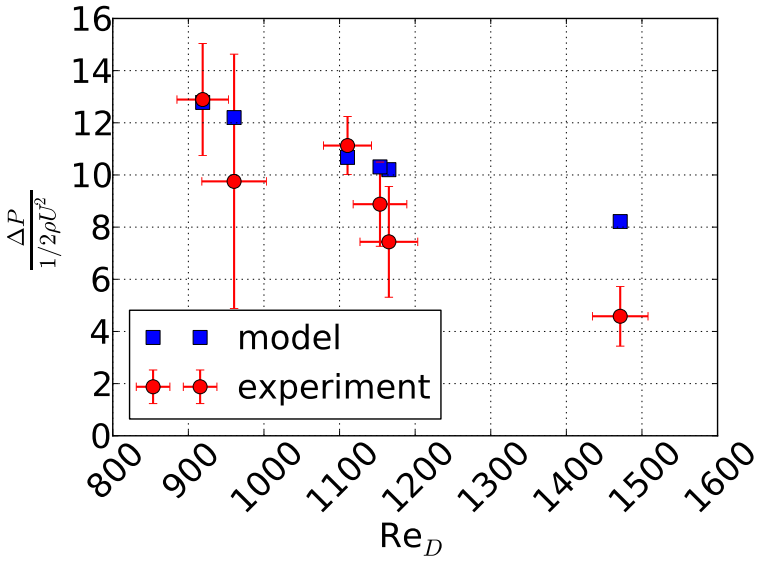


Figure 11.13: Plot of dimensionless pressure drop $\left(\frac{\Delta P}{1/2\rho U^2}\right)$ v. Re_D for model and experiment for copper mesh with fitted thermal resistance as TE surrogate material.

The correlation coefficients between the model and experimental data were 0.995 and 0.890 for Figures 11.12 and 11.13, respectively. Using thermal resistance as a fit parameter produced a good fit to the experimental heat transfer data. This was in spite of the fact that contact resistance, a parameter that is likely to vary with temperature, was assumed to be constant. Again, the fit of the pressure drop data was insignificantly affected because the overall heat exchanger effectiveness was small either way. As such, the predicted temperature profile and volume flow rate profile in the stream-wise direction did not change much, and therefore, pressure drop did not change much.

To reiterate, the thermal resistance determined from fitting the model to the experimental data was 8.21 m^2-K/kW, and the thermal resistance determined by the thermal resistance measuring device was 4.2 m^2-K/kW. Both of these were within the lower (25.4×10^{-3} m^2-K/kW) and upper (14.9 m^2-K/kW) bounds of the thermal

resistance predicted by Equations 11.1.1 and 11.1.2, though much closer to the upper bound. This is remarkably good agreement between the two values of thermal resistance considering that apart from thermal resistance, there are no free parameters in the model used for fitting. The value of the thermal resistance that was determined by the model is likely higher than the directly measured thermal resistance because the surfaces of the heat exchanger were not as uniformly flat over its extent so the contact resistance between the copper mesh and the walls of the heat exchanger was not spatially uniform. This would create areas of wider gaps and lower compression pressure than in the experimental test rig. Equation 11.1.1, Equation 11.1.2, nor direct measurement with the thermal resistance measurement device had any means to account for spatial variations in contact resistance over the contact surfaces of the heat exchanger.

11.3.3 Comparing the Surrogate Materials

In order to directly compare the experimental results for both materials, plots comparing the Cu mesh experimental results to the gypsum board experimental results are shown in Figure 11.14.

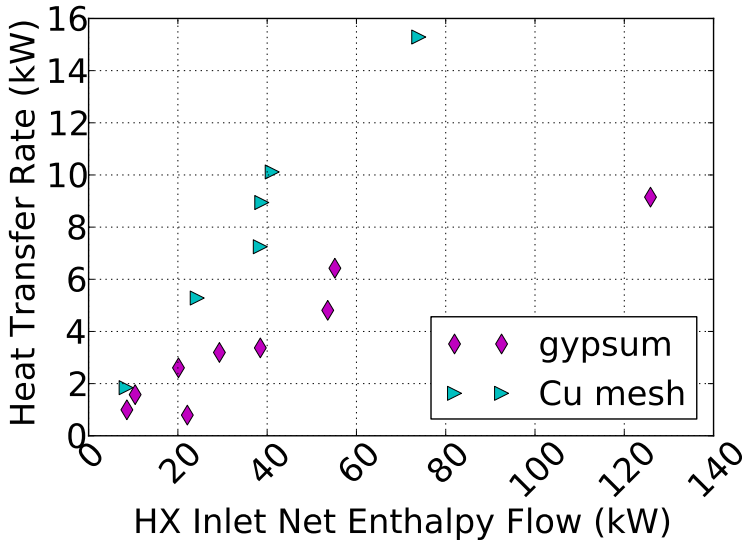


Figure 11.14: Plot of heat transfer rate v. net hot-side inlet enthalpy flow rate for both gypsum and copper mesh.

This plot shows that the heat transfer relative to net enthalpy flow was greater for the copper mesh, which had lower thermal resistance by a factor of 2.1. For reference, the effective thermal resistance of the optimized strip fin configuration was about 4.4 m^2-K/kW so the copper mesh was a good selection as a TE surrogate material.

11.4 Summary

Thermal resistance, which can be difficult to accurately determine [82], was used as a fit parameter to validate the convection model. Selecting the right value for thermal resistance resulted in good agreement between the heat transfer model and experimental heat transfer results over the entire range of flow and temperature conditions. This indicates that the convection model is accurate at predicting heat transfer if the total thermal resistance between the exhaust and coolant channels is

known.

In all pressure drop results there was a great deal of scatter in the experimental measurements. This was because some of conditions tested resulted in pressure drop as low as 0.1" water column. The precision of the manometer was 0.1 " water column so for these conditions, the uncertainy was high relative to the magnitude of the data.

The benefit of this second generation of experimental work is that it has demonstrated that the model is reasonably accurate at predicting both heat transfer and pressure drop for a thermoelectric waste heat recovery system. This partially validates the model results presented in Chapter 9, at least with respect to convection heat transfer and pumping work.

Chapter 12

Conclusions and Future Work

12.1 Conclusions

This modeling study used a numerical method to model the performance of a TE waste heat recovery system installed in the exhaust of a Cummins 6.7 L diesel engine. The model used a finite difference method to approximate the temperature and heat flux gradients in TE device legs as well as the temperature profile along the stream-wise flow direction in the system heat exchanger. This enabled the model to account for temperature and spatial variation of TE properties in both the stream-wise and transverse directions. In recognition of the fact that real TE devices are subject to convection boundary conditions, the model also used a numerical solver to determine the hot- and cold-side heat flux boundary conditions of the TE devices that satisfied the thermally asymmetrical heat flux boundary conditions between the TE devices and the hot- and cold-side fluids that arise from the internal energy conversion within the TE devices.

For all cases modeled, the heat exchanger system, consisting of TE elements mounted in a heat exchanger integrated into the exhaust of an engine, had a volume of 29.5 L, a system size typical of those being researched by others in the field, the length was 50.8 cm, the width was 50.8 cm, the exhaust duct height was 6.35 cm, the coolant duct height was 2.54 cm, and the exhaust flow enthalpy flux was 122 kW for an inlet temperature of 800 K and a mass flow rate of 7.9 kg/min. Temperature dependent material properties for $Mg_2Si_{0.5}Sn_{0.5}$ and $MnSi_{1.75}$ were calculated based

on second order polynomial fits of experimental data. The following heat exchanger internal configurations were considered in this model: empty ducts, exhaust duct filled with porous metal foam, exhaust duct lined with impinging manifold, exhaust and coolant ducts with strip fins, and exhaust duct with offset-strip fins and coolant duct with strip fins. For all configurations except the porous metal foams, a downhill simplex algorithm was employed to optimize TE leg length, TE n-/p-type leg area ratio, TE fill fraction, TE electrical current, and fin spacing for the offset-strip fins and strip fins.

For a heat exchanger with porous metal foams in the exhaust duct and strip fins in the coolant duct, the net power was negative due to excessive pressure drop, and because of this, no optimization was performed on the TE device parameters. For ducts without any heat transfer enhancement, the net power was 221.5 W. The optimum n-/p-type leg area ratio, leg length, current, and fill fraction were 0.703, 0.470 mm, 21.7 A, and 6.31 %, respectively. For exhaust ducts with impinging jets, the net power was 499 W. The optimum n-type to p-type n-/p-type leg area ratio, leg length, current, and fill fraction were 0.702, 0.579 mm, 17.8 A, and 1.77 %, respectively. For coolant ducts with strip fins and exhaust ducts with offset-strip fins, the net power was 2.11 kW. The optimum n-type to p-type n-/p-type leg area ratio, leg length, current, and fill fraction were 0.700, 2.57 mm, 4.13 A, and 32.0 %, respectively. The optimal offset-strip fin spacing was 2.12 mm. For coolant and exhaust ducts with strip fins, the net power was 2.25 kW. The optimum n-type to p-type n-/p-type leg area ratio, leg length, current, and fill fraction were 0.700, 3.00 mm, 3.61 A, and 37.3 %, respectively. The importance of this four parameter design space cannot be over-emphasized. In particular, if leg length, fill fraction, or current is not properly chosen in the design, the TE device efficiency drops rapidly, and even small deviations from optimal values can result in resistance heating rather than

energy conversion. These optimal parameters are also highly sensitive to boundary conditions.

For the first generation experimental work, a small-scale 1.1 liter volume flat plate heat exchanger was fabricated to study the performance characteristics of a conceptual design for waste heat recovery in diesel exhaust. The heat exchanger consisted of an exhaust channel and two coolant channels all having rectangular cross-sections. The experimentally measured heat transfer rates were compared with a heat transfer model to be used both for heat exchanger development and for modeling thermoelectric device performance. In both the model and the experiment, alumina paper was used as a surrogate for the thermoelectric materials. The heat exchanger was modeled using a finite element method to accommodate temperature-dependent material properties. The minimum and maximum heat transfer rates calculated from the model were 188 W and 1.89 kW, respectively, and the heat exchanger effectiveness from the model ranged from 0.110 to 0.169 for gas exchange rates from 65.7 s^1 to 334 s^1. The measured minimum and maximum heat transfer rates from the experiments were 385 W and 1.45 kW, with effectiveness ranging from 0.079 to 0.258.

The model, which contained no free parameters, was successful at reproducing the approximate trends of the experiments, but differences in heat transfer rates as great as 40 % were noted for the highest and lowest flow rates. The model predicted that the heat transfer rate would increase approximately linearly with hot-side flow rate, whereas the measured heat transfer rate increased at a diminishing rate as flow rate increased.

While the reason or reasons for this could not be known with complete certainty, possible explanations for why the model and experimental heat transfer rates were different were hypothesized. Unaccounted for heat transfer from the un-insulated

triangular end caps would contribute to a greater measured temperature drop than if the end caps had been insulated, and thus yield a measured heat transfer rate greater than predicted by the model. This effect would be consistent with the under-predicted heat transfer rate at the lower flow rates. The shorter residence time of the flow through the end- caps at higher flow rates would result in less time for heat transfer, consistent with diminishing heat transfer rates, if the convective heat transfer coefficient did not increase proportionately. The entrance flow to the triangular end cap resembles an impinging jet. It is possible that this could have contributed to relatively high heat transfer losses from the end cap. The scaling of this impingement effect with increasing flow rate in unknown and creates another uncertainty. In any future measurements, the end caps will be insulated or the thermocouples will be moved to locations within the main body of the heat exchanger.

For the second generation of experimental work, a much larger heat exchanger that could accommodate the entire flow of the Cummins exhaust was built. At this point in the research, thermoelectric devices were still not available so gypsum board and copper mesh were selected as two representative TE surrogate materials. These materials spanned a range of thermal resistance in order to emulate a range of possible TE device designs and/or materials. For the gypsum board, the model under-predicted the heat transfer, though the trend-wise agreement was good. The flow model slightly over-predicted the pressure drop, though there was substantial scatter in the experimental data. For the copper mesh as a TE surrogate material, heat transfer was over-predicted, but again, the trend-wise agreement was good. The pressure drop was more accurately predicted for this TE surrogate material. To determine whether the convection model itself, or perhaps simply the assumed thermal resistance, was resulting in the error in predicting the experimental results, the thermal resistance of both TE surrogate materials was used as a fit parameter in a least

squares analysis. This resulted in much better agreement between the heat transfer model and experimental data for both the gypsum board and the copper mesh TE surrogate materials. The pressure drop model was not significantly changed by using this fit parameter.

The benefit of this second generation of experimental work is that it has demonstrated that the model is reasonably accurate at predicting both heat transfer and pressure drop for a thermoelectric waste heat recovery system. In fact, the model was much more accurate for predicting the second generation heat exchanger heat transfer and pressure drop compared to these same predictions for the first generation. The tendency to under-predict heat transfer at low flow rates and over-predict heat transfer at high flow rates that was exhibited by comparing the model results with the first generation experimental results was not present in the second generation experiment. The agreement between the model and the second generation experimental results partially validates the previously discussed model optimization results, at least with respect to convection heat transfer and pumping work. This demonstrates that the model can be used as a design tool for thermoelectric waste heat recovery systems in automotive applications. Because the net power output is extremely sensitive to optimal selection of parameters in the design space and sensitive to operating conditions, and the design space itself is extremely sensitive to operating conditions, the successful use of TE devices for automotive exhaust waste heat recovery is highly dependent on a highly adaptable, controllable TE system design.

12.2 Future Work

The most obvious improvements for this work involve thoroughly addressing all of the issues involved in optimizing TE waste heat recovery systems for operation over

a drive cycle. A good starting point for this goal is to plot optimum TE parameters as a function of varied exhaust flow rate and inlet temperature. This would provide a means of determining how well a single set of optimized TE design space parameters can fit a drive cycle with infinitely fast response to changing conditions. If this produces a result that indicates that no single set of optimal parameters can produce an acceptable level of power output over a drive cycle, then it may be unnecessary to proceed beyond this.

Otherwise, it will be advisable to develop a transient model because the characteristic response time for the heat exchanger system will likely be on the order of tens of minutes. The transient model can then be used to verify the existence of a single set of optimal design space parameters that will produce an acceptable power output over a drive cycle with transient response to changes in operating condition. This is important because there is no guarantee that the same optimal parameters found by assuming infinitely fast transient response will produce the same result when a realistic time response is modeled.

There are some minor improvements that can be made to this work to extend the usefulness of the modeling. First, it would be a good idea to modify the TE model to specify electrical load resistance and then iteratively solve for electrical current. Load resistance would be easier to hold constant in experiments with varied conditions so the model will be more useful for matching experimental data with actual TE devices if this improvement is made. Second, it would be a good idea to explore the relationship between optimal current and TE leg length per fill fraction. Length per fill fraction may be a useful dimensional parameter, and there may even be a related dimensionless parameter that can be used to represent this effectively. Third, it may be the case that the optimal TE leg length is not realistic for manufacturing

purposes, and if this turns out to be the case, it would be a good idea to understand what the optimal set of design space parameters is with a constraint on the TE leg length.

Perhaps a drastic change to the direction of this work would be to explore the use of a phase change heat exchanger between the exhaust and the TE hot side with a super-heated working fluid in a circular flow. By having a phase change heat exchanger utilizing a loop rather than having both phases present in the same channel connecting the hot and cold side, the working fluid could be super-heated by the exhaust flow before coming in contact with the TE devices.

It would probably also be worthwhile to compare TE waste heat recovery techniques such as turbo-compounding, in which a turbine converts exhaust enthalpy and kinetic energy into useful shaft work. The complexity of the TE system for automotive use suggests that this application is probably not a good match for TE devices. Optimizing for even a single operating condition requires at least four parameters that are not guaranteed to be optimal over a range of operating conditions. Add to this the fact that automobiles not only operate over a range of condition, but also that changes between operating conditions are highly transient and frequent, and it is clear that applying TE waste heat recovery to automobiles needs to be approached with a healthy level of skepticism.

Appendices

Appendix A

Calculation of Transport Properties

All transport properties for both projects were calculated using methodologies and data from Transport Phenomena by Bird *et al.* [41]. Some data were also taken from an undergraduate thermodynamics text book [83].

For the calculation of dynamic viscosity, the following formula (equation 1.4-14 from Bird *et al.*[41]) was used:

$$\mu = \frac{5}{16}\frac{\sqrt{\pi m k_B T}}{\pi d^2} \tag{A.0.1}$$

where μ is dynamic viscosity, m is the molecular mass, k_B is Boltzmann's constant, T is the temperature of the gas, and d is the collision diameter of the gas molecule. Table A.1 shows the values used for various gases, all taken from Bird *et al.*

Table A.1: Molecular properties used for bulk property calculation.

Gas	Collision Diameter (Å)	Molecular Mass (kg)	Pr
Air	3.617	28.964	0.74
Propane	4.934	44.10	-

For thermal properties, the specific heat and Prandtl number (Pr) are needed. For this work, these properties were needed for air only. The Prandtl number is provided in Table A.1, and the 4th order polynomial used for calculating the constant pressure specific heat of air is [83]:

$$c_{p,\text{air}} = \sum_{i=0}^{4} C_i T^i \tag{A.0.2}$$

where the coefficients C_i are provided in Table A.2.

Table A.2: Polynomial coefficients used for calculating $c_{p,air}$.

C_0	C_1	C_2	C_3	C_4
3.653	$1.337*10^{-3}$	$3.294*10^{-6}$	$1.913*10^{-9}$	$0.2763*10^{-12}$

When the specific heat, $c_{p,\text{air}}$, and dynamic viscosity, μ, are both known, the thermal conductivity can be calculated using

$$k = \alpha \rho c_{p,\text{air}} \tag{A.0.3}$$

where α is the thermal diffusivity, calculated using

$$\alpha = \frac{\mu}{\rho Pr} \tag{A.0.4}$$

and ρ is the ideal gas density of air.

Appendix B

Uncertainty Analysis for 2nd Generation Heat Exchanger

This Appendix will discuss uncertainty in pressure drop and heat transfer for the second generation heat exchanger experiments.

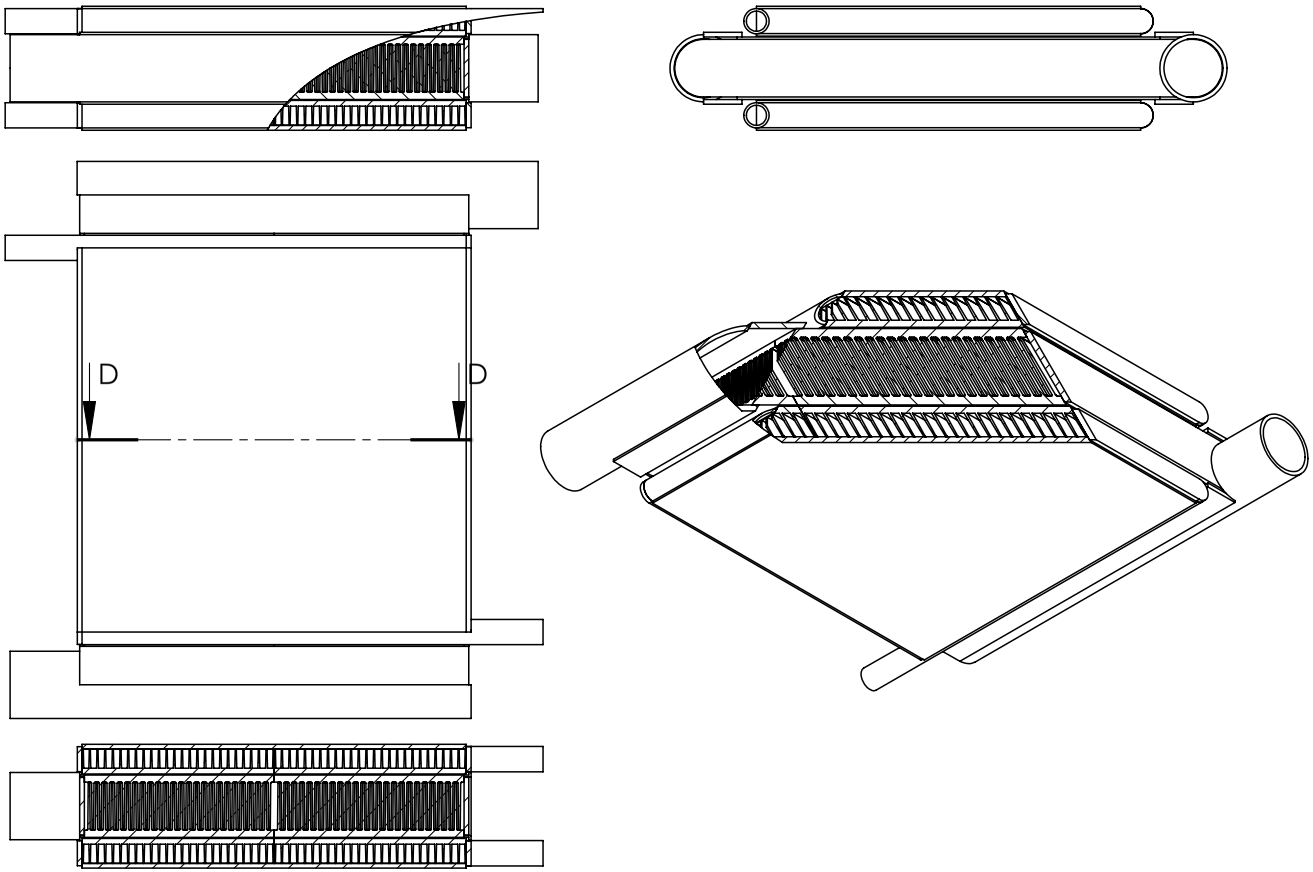
The uncertainty for the pressure drop was due to precision of the manometer used to measure the change in water column across the exhaust side of the heat exchanger. This value was 0.2" water column, which converts to 49.8 Pa. The error bars in Figures 11.7, 11.9, 11.11, and 11.13 reflect this uncertainty.

For the exhaust mass flow rate, the uncertainty was estimated to be 0.2 kg/min based on observed fluctuations, and for the exhaust temperature measurements, the uncertainty was 2.2 °C or 0.75 %, whichever is greater [84]. These uncertainties impacted the Reynolds number, the heat transfer, and the net enthalpy flow, and they are reflected in error bars on all of these variables in Section 11.3.

The uncertainties of specific heat and dynamic viscosity were assumed to be insignificant compared to the uncertainties in temperature and mass flow rate.

Appendix C

Engineering Drawings



SECTION D-D

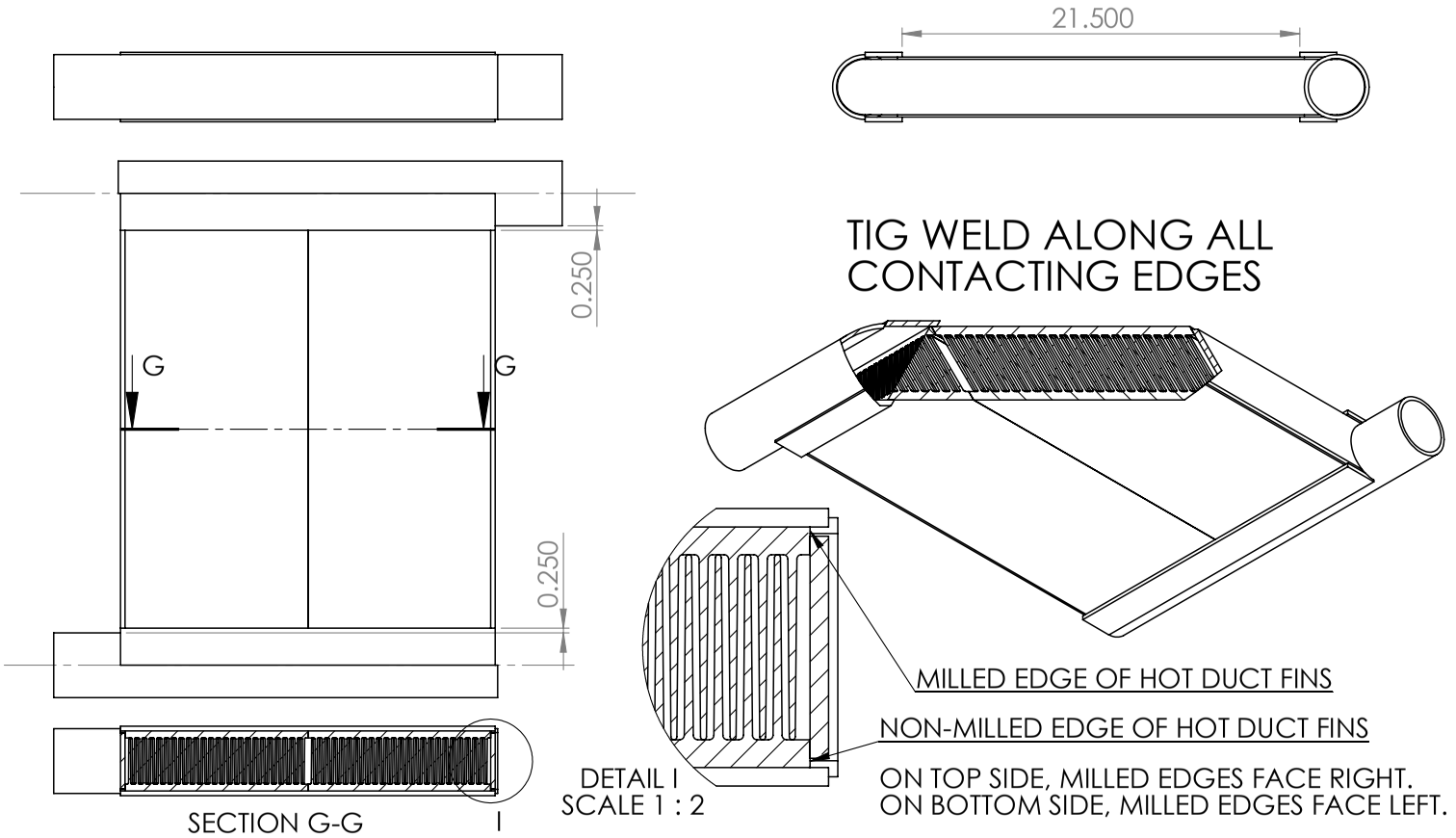
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

HX ASSEMBLY

SIZE	DWG. NO.	REV
A	hx	
SCALE: 1:8	WEIGHT:	SHEET 1 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

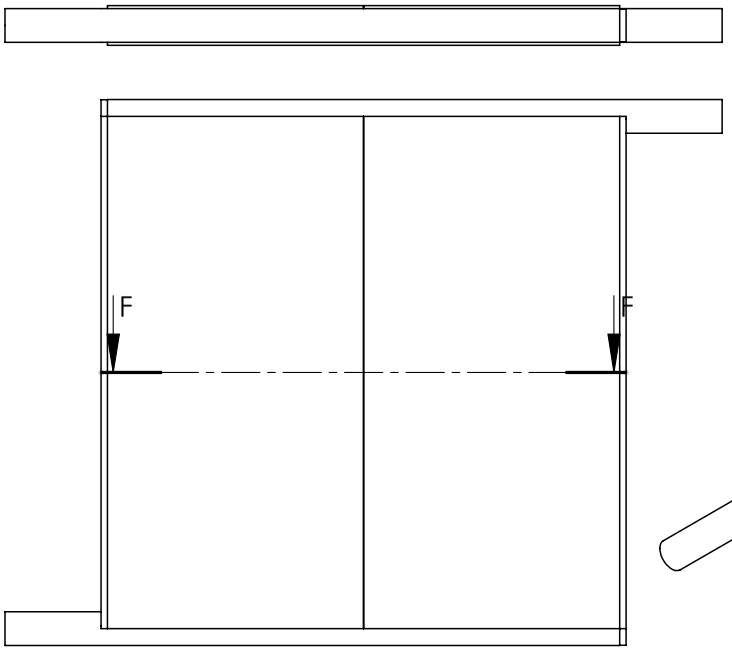
TITLE:

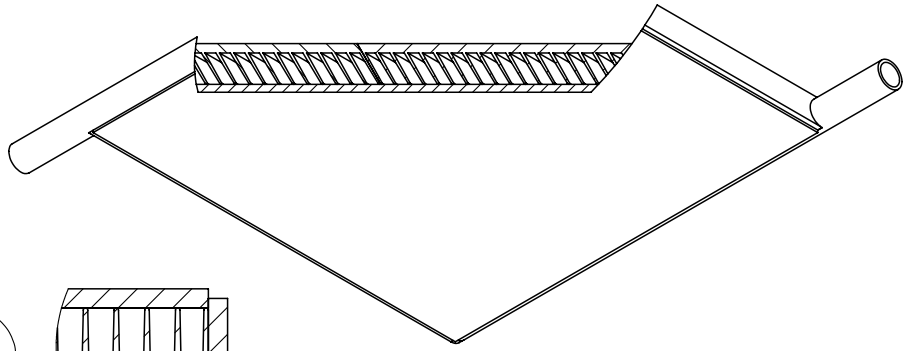
HOT DUCT

SIZE	DWG. NO.	REV
A	hx	

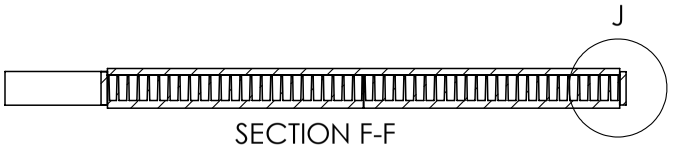
SCALE: 1:8 | WEIGHT: | SHEET 2 OF 12

5 4 3 2 1





TIG WELD ALONG ALL CONTACTING EDGES



SECTION F-F

DETAIL J
SCALE 1 : 2

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

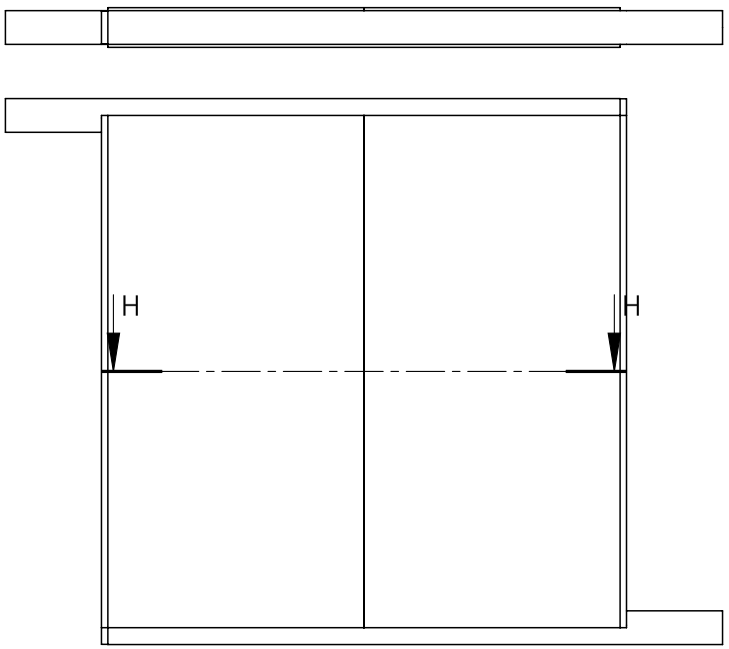
TITLE:

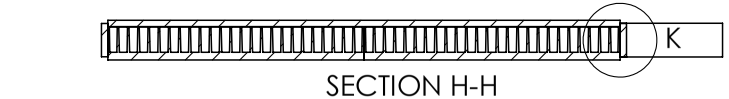
COLD DUCT

SIZE	DWG. NO.	REV
A	hx	

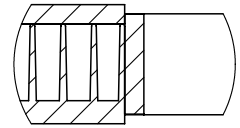
SCALE: 1:6 | WEIGHT: | SHEET 3 OF 12

5 4 3 2 1



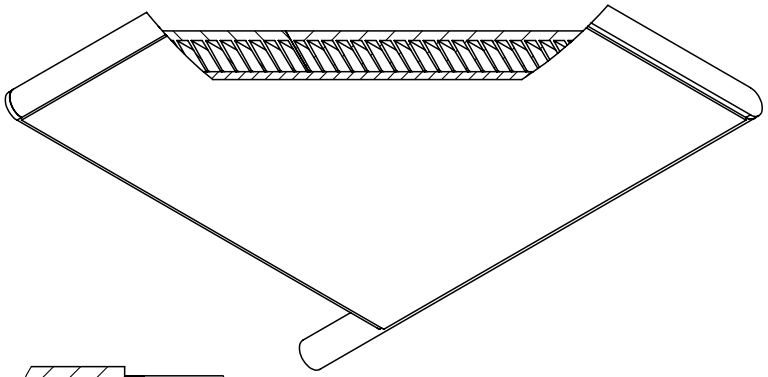


SECTION H-H



DETAIL K
SCALE 1 : 2

TIG WELD ALONG ALL CONTACTING EDGES



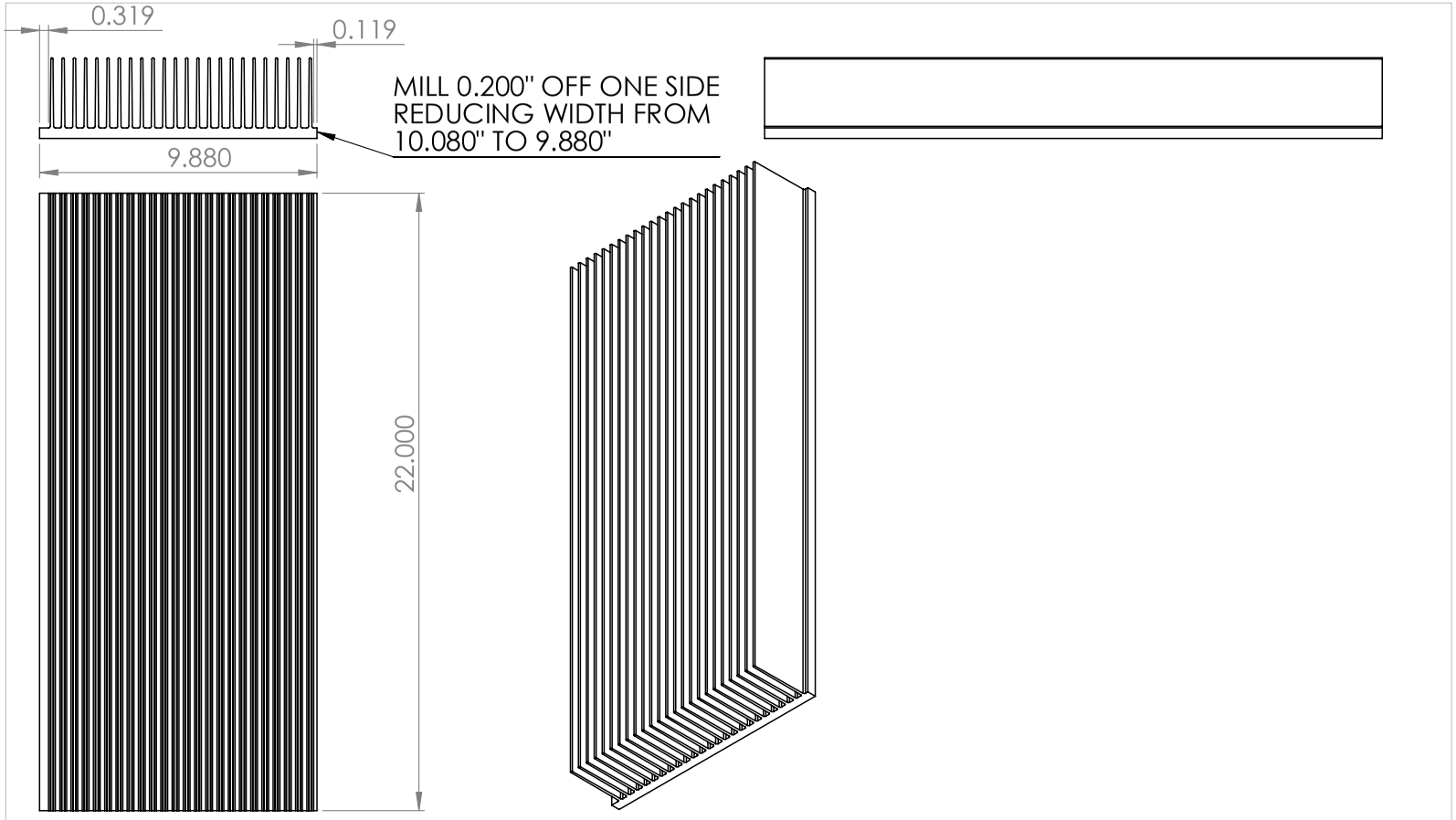
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE: COLD DUCT MIRRORED

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:6 | WEIGHT: | SHEET 4 OF 12

5 4 3 2 1



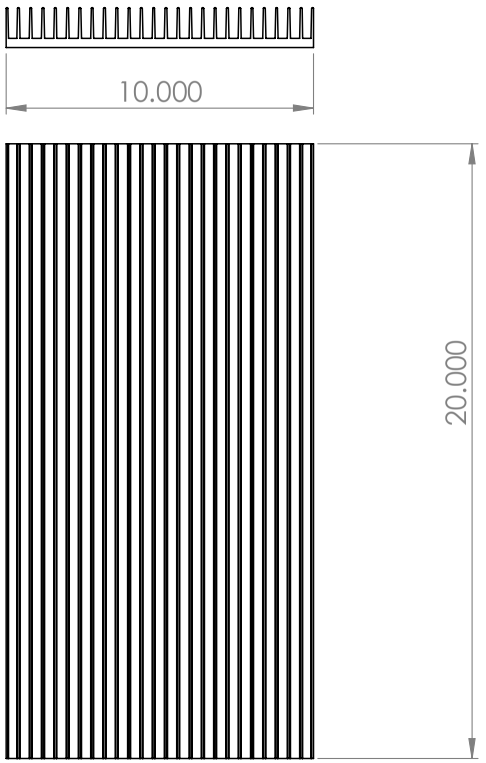
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS: QTY. 4		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

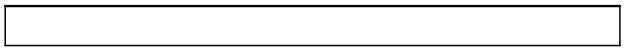
TITLE: HOT DUCT FINS

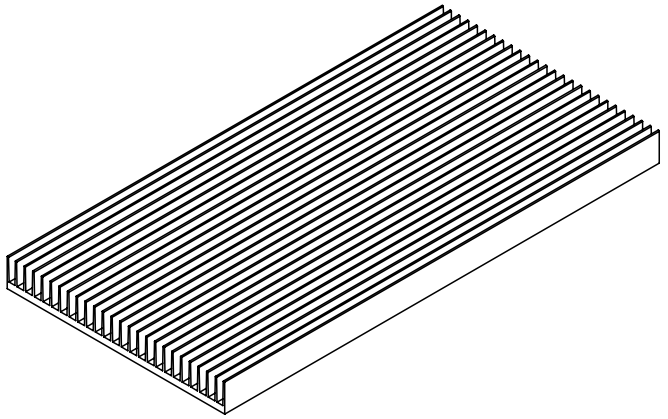
SIZE A | DWG. NO. hx | REV

SCALE: 1:5 | WEIGHT: | SHEET 5 OF 12

5 4 3 2 1







		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 4		
APPLICATION		DO NOT SCALE DRAWING			

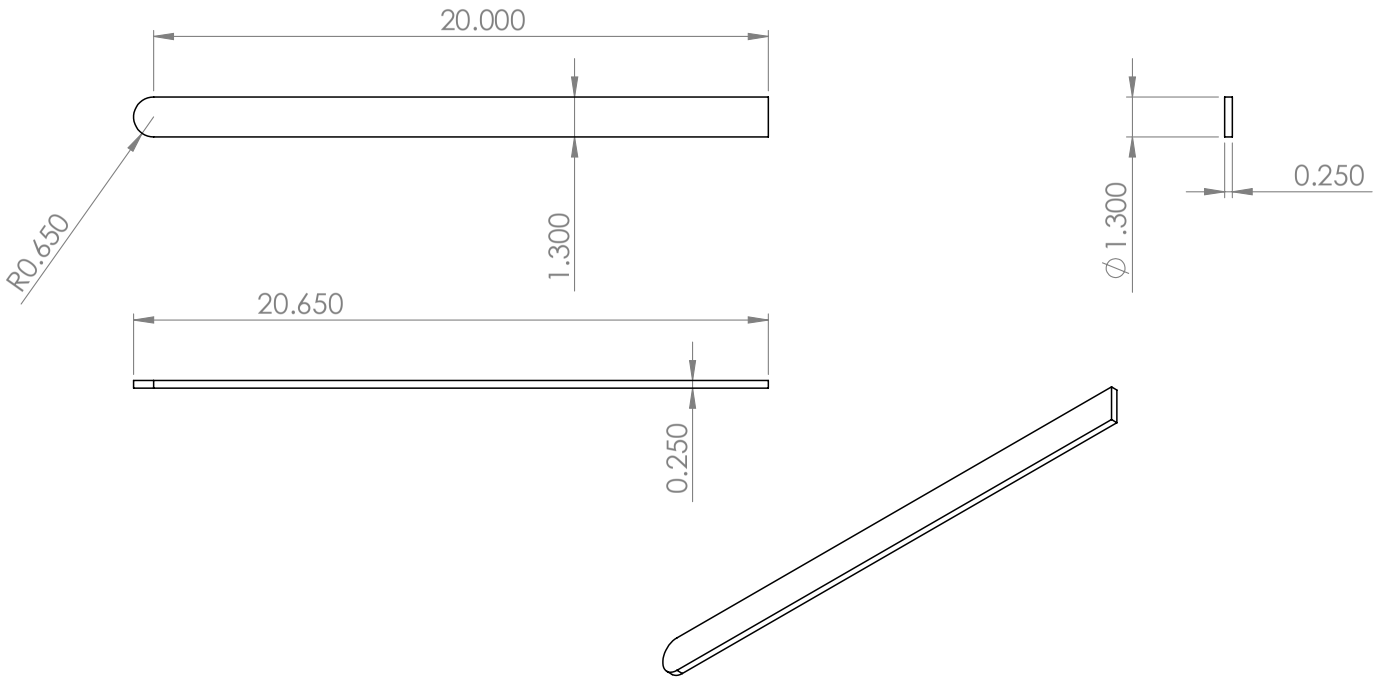
TITLE:

COLD DUCT FINS

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 6 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 4		
APPLICATION		DO NOT SCALE DRAWING			

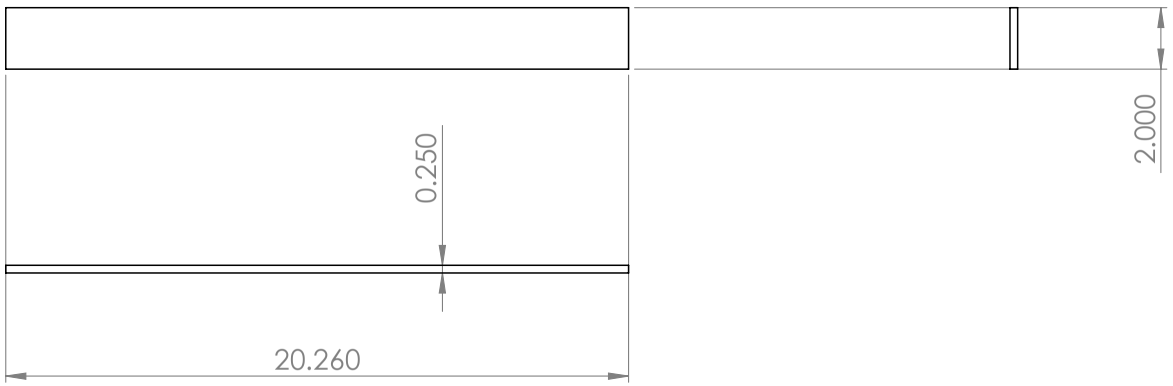
TITLE:

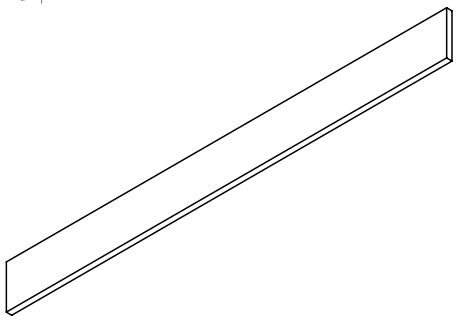
COLD SIDE PLATE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 7 OF 12

5 4 3 2 1





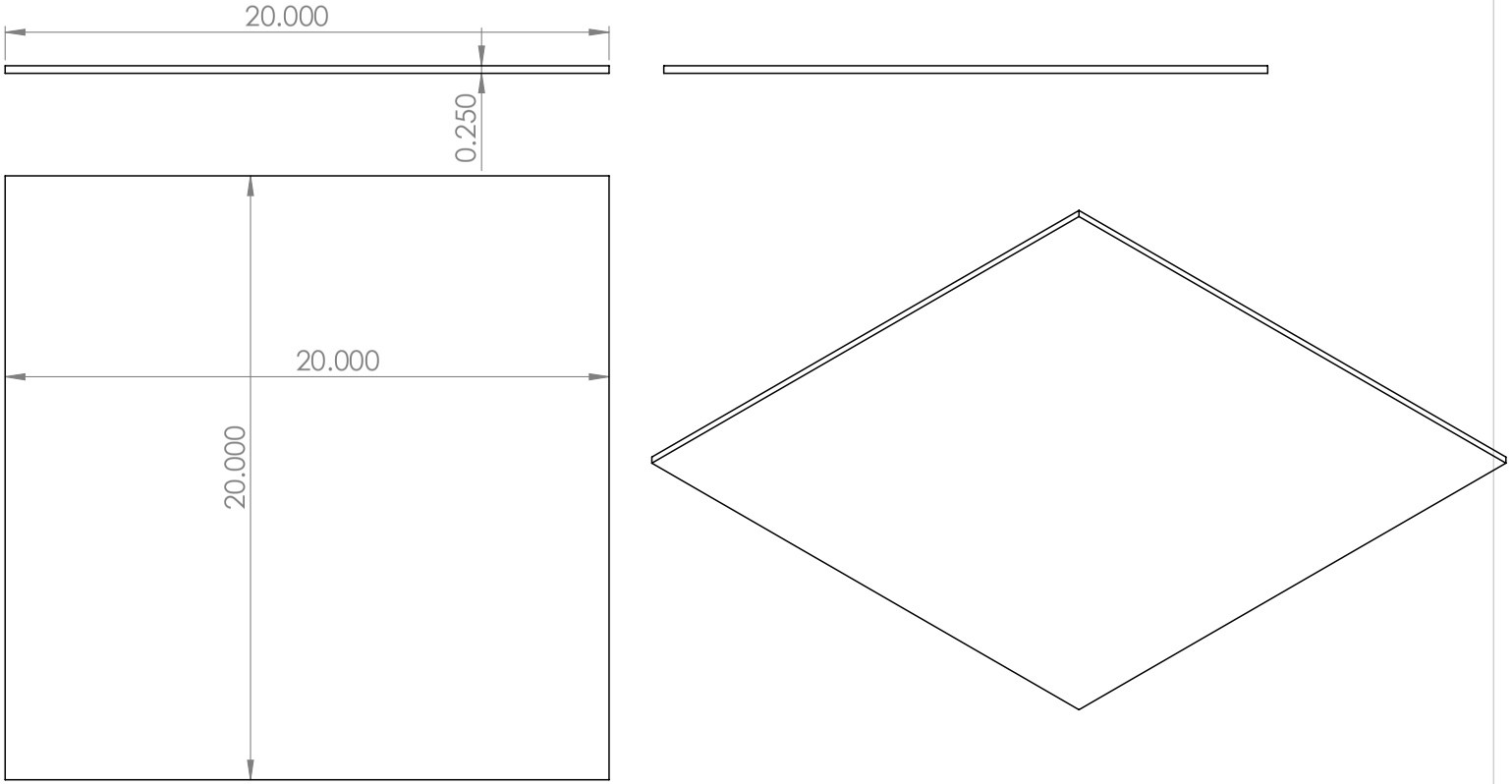
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 2		
APPLICATION		DO NOT SCALE DRAWING			

TITLE: HOT EXTENSION PLATE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 8 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 2		
APPLICATION		DO NOT SCALE DRAWING			

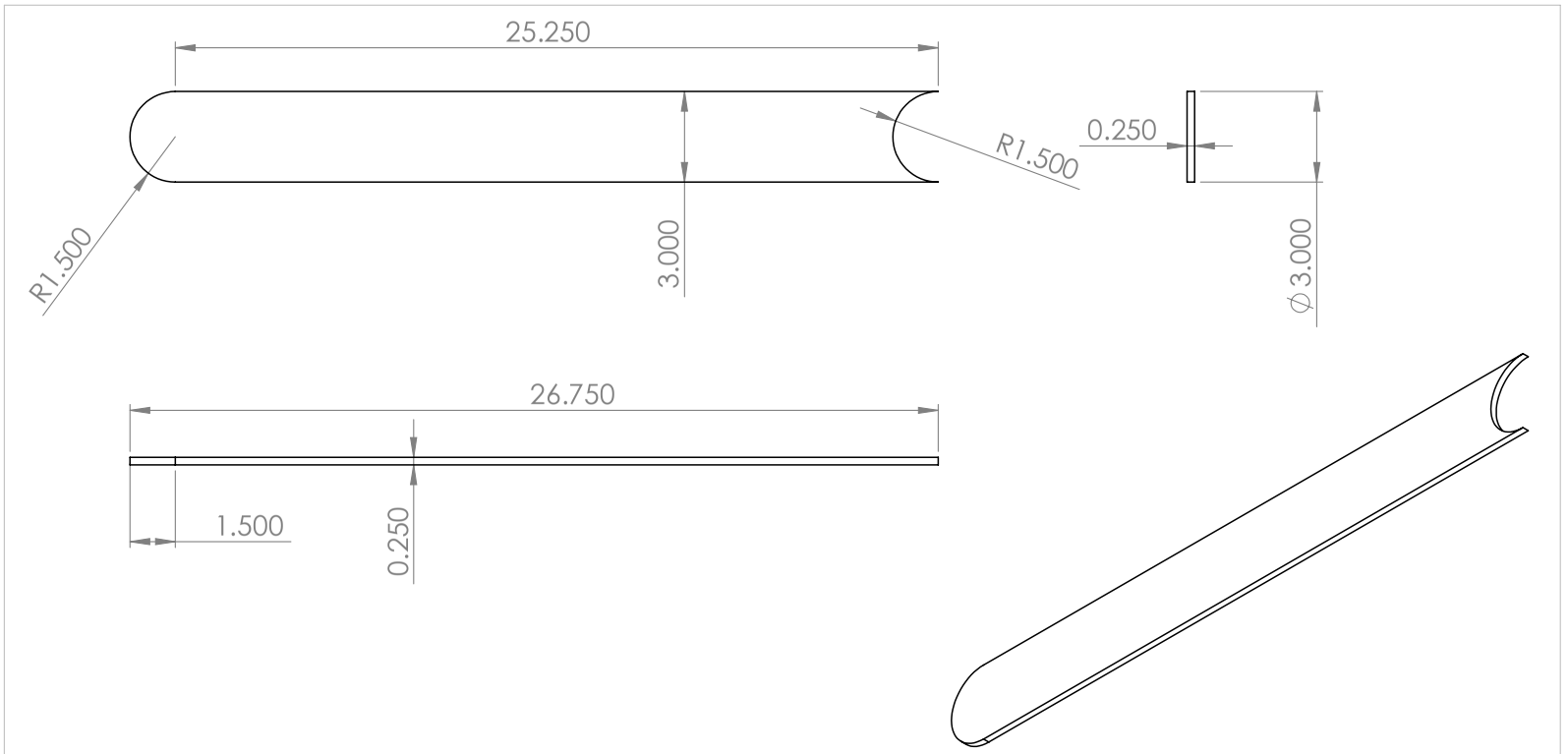
TITLE:

COLD TOP PLATE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 9 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 2		
APPLICATION		DO NOT SCALE DRAWING			

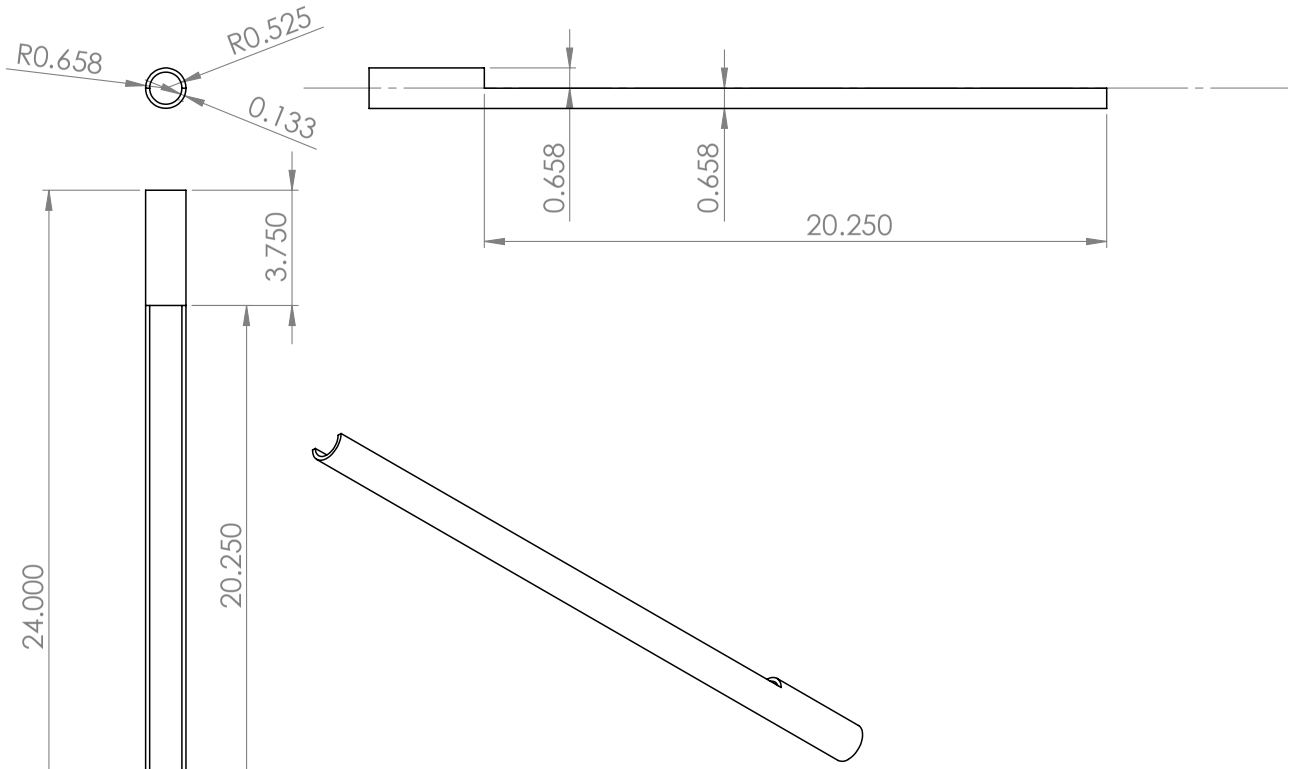
TITLE:

HOT SIDE PLATE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 10 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 4		
APPLICATION		DO NOT SCALE DRAWING			

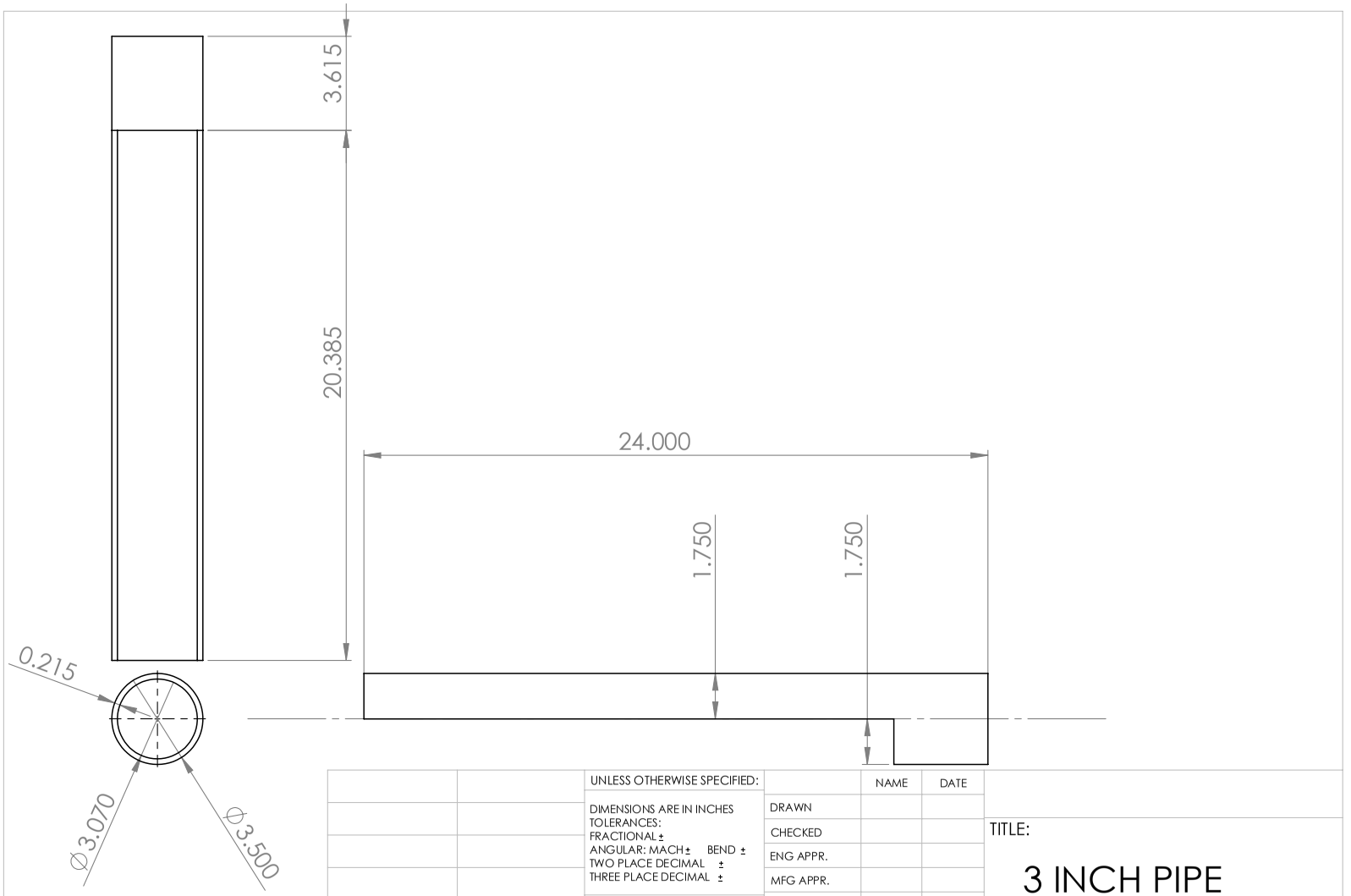
TITLE:

1 INCH PIPE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 11 OF 12

5 4 3 2 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH	QTY. 2		
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

3 INCH PIPE

SIZE	DWG. NO.	REV
A	hx	

SCALE: 1:5 | WEIGHT: | SHEET 12 OF 12

5 4 3 2 1

Appendix D

Cummins Operation

This chapter explains everything needed to operate the Cummins engine for the purpose of duplicating the results in this dissertation. For any additional information about operating and troubleshooting the Cummins operating interface, please consult Appendix C of Tim Diller's dissertation [85].

D.1 Setting up Calterm III

- Contact Michael Aikins (michael.a.aikins@cummins.com) at Cummins to get a license file.
- Follow his installation instructions.
- Follow instructions for selecting a module from Appendix C of Tim Diller's Dissertation [85].
- To set up the communication link with the engine, in Calterm III, go to Tools > Options > Datalink and select Peak_systems for Adapter.
- For adding all the particular sensor outputs you like, hit F1 in the data output screen, and there is a search feature.

D.2 Running the Engine

This section will explain what tasks need to be done in order to run the engine. Some tasks need to be performed only occasionally, and some need to be performed every time the engine is run.

D.2.1 Occasional Tasks

- Periodically check the oil level
- Periodically check the coolant level
- Periodically change the oil
- Periodically change the coolant

D.2.2 Startup Tasks

Execute this checklist every time the engine is run:

1. Open the valve to turn on the dyno cooling water
2. Check the fuel level. Fill if the tank is less than about half full because you need some fuel in the tank to make sure it doesnt overheat after coming out of the return line. The fuel tank cap does not thread properly. This is okay.
3. Open the fuel valve.
4. If the heat exchanger is installed in the exhaust, open the heat exchanger coolant valve.
5. Turn on the air vent for the lab.
6. Check that the battery circuit is closed.

7. Turn on the dyno by pressing the "POWER ON" button on the Super Flow SF-901 control panel.

8. Verify that the "Speed Control" knob is all the way counter clockwise for idle.

9. Verify that the "LOAD" knob is all the way clockwise for minimum load.

10. Verify that the "Tip-in" toggle switch is in the up (off) position. Leaving this down during engine start will violently rev the engine, potentially causing damage.

11. Turn on the intercooler fans using the two three-way toggle switches. Middle is off. Up or down is on.

12. Turn on the ignition (red light should come on).

13. Press the starter button.

14. Keep it under 1200 rpm until coolant temperature (see Calterm for this) is above $\sim$75 °C. Also, keep the load below 100 ft-lb until this condition is met.

D.3 Miscellaneous Instructions

- Always reduce engine speed before reducing load.

- In case of emergency, the two emergency buttons (one on the dyno and one on the intercooler) can be used for a hard, immediate shutoff.

- The hard shutoff be avoided for minor problems because this will shorten the life of the fuel injectors. This is because the fuel injectors rely on continuous operation for lubrication and cooling. The preferred way to kill the engine is

to reduce the Speed Control knob all the way to zero (counter clockwise) and then flip the ignition toggle switch down.

- If the LED lights on the dyno show erratic numbers and the ignition toggle switch does not light up in the up (on) position, press the "STOP PROGRAM" button to make sure the dyno is not running a program. This should fix the problem.

Appendix E

Python Code

In this Chapter, code that is specific to the catalyst project (Part I) will be presented in Section E.1, code that is specific to the waste heat recovery project (Part II) will be presented in Section E.2, and code that is common to both projects will be presented in Section E.3.

For a complete revision history for all codes, please see my github account page: https://github.com/calbaker.

E.1 Catalyst Code

All code in this section can be found here:

https://github.com/calbaker/Catalyst.

Listing E.1: catalyst.py

```
"""Contains class definition for catalyst."""

# Distribution libraries
import numpy as np
import types

# Local libraries
import properties as prop
reload(prop)

import analytical
reload(analytical)
import experimental
reload(experimental)
import numerical
reload(numerical)
import prop_functions
reload(prop_functions)

class Catalyst(object):

    """Class for representing catalyst reactor.
```

```
Reactor can modeled by 1 to 4 term Fourier expansion.

A word on units:
Pressure is always in kPa unless otherwise specified
Temperature ditto K ditto
Lengths ditto m ditto"""

def __init__(self, **kwargs):
    """Sets values of constants"""

    self.init_analytical()
    self.init_numerical()
    self.init_prop_functions()
    self.init_experimental()

    self.P = 101.325  # Pressure of flow (kPa)

    self.lambda_and_Da = np.array(
        [
            [1e-5, 3.16e-03, 3.14e+00, 6.28e+00, 9.42e+00,
        1.25e+01, 1.57e+01, 1.88e+01, 2.19e+01, 2.51e+01,
        2.82e+01],
            [1e-4, 9.99e-03, 3.14e+00, 6.28e+00, 9.42e+00,
        1.25e+01, 1.57e+01, 1.88e+01, 2.19e+01,  2.51e+01,
        2.82e+01],
            [1e-3, 0.03, 3.14, 6.28, 9.42, 12.6, 15.7, 18.8, 22.0,
        25.1, 28.3],
            [2e-3, 0.044, 3.14, 6.30, 9.42, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [3e-3, 0.055, 3.14, 6.28, 9.43, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [4e-3, 0.063, 3.14, 6.28, 9.43, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [5e-3, 0.071, 3.14, 6.28, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [6e-3, 0.0773, 3.14, 6.28, 9.43, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [7e-3, 0.0836, 3.14, 6.28, 9.43, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [8e-3, 0.0893, 3.14, 6.28, 9.43, 12.6, 15.7, 18.8,
        22.0, 25.1, 28.3],
            [9e-3, 0.0947, 3.14, 6.28, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.01, 0.010, 3.14, 6.28, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.015, 0.122, 3.15, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.02,  0.141, 3.15, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.025, 0.157, 3.15, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.03,  0.172, 3.15, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.2],
            [0.04, 0.199, 3.15, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.05, 0.222, 3.16, 6.29, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.09, 0.300, 3.17, 6.30, 9.43, 12.6, 15.7, 18.9,
        22.0, 25.1, 28.3],
            [0.1, 0.311, 3.17, 6.30, 9.44, 12.6, 15.7, 18.9, 22.0,
        25.1, 28.3],
            [0.11, 0.326, 3.18, 6.30, 9.44, 12.6, 15.7, 18.9,
```

```
    22.0, 25.1, 28.3],
        [0.2,  0.433, 3.20, 6.31, 9.45, 12.6, 15.7, 18.9,
    22.0, 25.1, 28.3],
        [0.3,  0.522, 3.23, 6.33, 9.46, 12.6, 15.7, 18.9,
    22.0, 25.1, 28.3],
        [0.4, 0.593, 3.26, 6.35, 9.47, 12.6, 15.7, 18.9, 22.0,
    25.1, 28.3],
        [0.5, 0.653, 3.29, 6.36, 9.48, 12.6, 15.7, 18.9, 22.0,
    25.2, 28.3],
        [1.0, 0.860, 3.43, 6.44, 9.53, 12.6, 15.8, 18.9, 22.0,
    25.2, 28.3],
        [5.,  1.31, 4.03, 6.91, 9.89, 12.9, 16.0, 19.1, 22.2,
    25.3, 28.4],
        [10.0,  1.43, 4.31, 7.23, 10.2, 13.2, 16.3, 19.3,
    22.4, 25.5, 28.6]
        ]
    )
# Graphically determined eigenvalues corresponding to Da.
# First column is Da, second column is lamba_0, third column
# is lambda_1, and so on...

self.init_lambda_splines()

if 'terms' in kwargs:
    self.terms = kwargs['terms']
else:
    self.terms = 4

self.A_arr = 2.015e5
# Arrhenius coefficient (1/s)
self.T_a = 5.739e3  # activation temperature (K)

# Nanowire morphology
self.porosity = 0.97  # porosity of nanowires
self.tortuosity = self.porosity ** -1
# tortuosity of nanowires
self.Kn_length = 100e-9
# Knudsen length (m) scale
self.thickness = 5.e-6
# Thickness of wash coat or height of porous media (m).  This
# was h_{pore} in the pdf.

# Channel geometry
self.height = 0.0025
# channel height (m)
self.length = 76.2e-3 * 2.
# channel length (m)
self.width = 20e-3  # channel width (m)

self.x_ = self.length / self.height
# dimensionless x.  this will need to be reevaluated if length
# or height is changed.

self.y_ = 1.

self.x_array = np.linspace(0, self.x_, 100)
self.y_array = np.linspace(0, self.y_ * 51 / 50, 50)

self.T_ambient = 300. + 273.15
# ambient temperature (K) at which flow rate is measured

self.fuel = prop.ideal_gas(species='C3H8')
self.air = prop.ideal_gas()
```

```
    self.air.P = self.P
    self.fuel.P = self.P

def init_analytical(self):

    """Adds methods for analytical model."""

    self.init_lambda_splines = (
        types.MethodType(analytical.init_lambda_splines, self)
        )
    self.set_eta_ij = types.MethodType(analytical.set_eta_ij, self)
    self.get_eta = types.MethodType(analytical.get_eta, self)
    self.get_eta_fit = types.MethodType(
        analytical.get_eta_fit, self
        )
    self.get_Y = types.MethodType(analytical.get_Y, self)
    self.get_A_i = types.MethodType(analytical.get_A_i, self)
    self.get_lambda_spl = (
        types.MethodType(analytical.get_lambda_spl, self)
        )
    self.get_lambda_error = (
        types.MethodType(analytical.get_lambda_error, self)
        )
    self.get_lambda = types.MethodType(analytical.get_lambda, self)


def init_numerical(self):

    """Adds methods from numerical."""

    self.solve_numeric = types.MethodType(numerical.solve_numeric,
    self)
    self.set_eta_ij_num = (
        types.MethodType(numerical.set_eta_ij_num, self)
        )
    self.get_eta_num = types.MethodType(numerical.get_eta_num,
    self)
    self.get_Yprime = types.MethodType(numerical.get_Yprime, self)


def init_prop_functions(self):

    """Adds methods from prop_functions."""

    self.get_Da = types.MethodType(prop_functions.get_Da, self)
    self.get_thiele = types.MethodType(prop_functions.get_thiele, self)
    self.get_k = types.MethodType(prop_functions.get_k, self)
    self.get_Pe = types.MethodType(prop_functions.get_Pe, self)
    self.set_TempPres_dependents = (
        types.MethodType(prop_functions.set_TempPres_dependents, self)
        )
    self.get_mfp = types.MethodType(prop_functions.get_mfp, self)
    self.get_Kn = types.MethodType(prop_functions.get_Kn, self)
    self.get_D_C3H8_air_eff = (
        types.MethodType(prop_functions.get_D_C3H8_air_eff, self)
        )
    self.get_D_C3H8_air_Kn = (
        types.MethodType(prop_functions.get_D_C3H8_air_Kn, self)
        )
    self.get_D_C3H8_air = (
        types.MethodType(prop_functions.get_D_C3H8_air, self)
        )
```

```
    def init_experimental(self):

        """Adds methods from experimental."""

        self.get_S_r = types.MethodType(experimental.get_S_r, self)
        self.import_data = types.MethodType(
            experimental.import_data, self
            )
        self.set_fit_params = types.MethodType(
            experimental.set_fit_params, self
            )
```

Listing E.2: analytical.py

```
import numpy as np
import scipy.interpolate as interp
from scipy.optimize import fsolve

def init_lambda_splines(self):

    """Sets up spline fitting for get_lambda."""

    self.lambda_splines = []

    max_terms = self.lambda_and_Da.shape[1] - 1

    for i in range(0, max_terms):
        self.lambda_splines.append(
            interp.splrep(self.lambda_and_Da[:, 0],
                          self.lambda_and_Da[:, i + 1])
            )
    self.lambda_splines = np.array(self.lambda_splines)

def set_eta_ij(self):

    """Sets conversion efficiency over a range of Pe and Da."""

    self.x_ = self.length / self.height

    try:
        self.Vdot_array
    except AttributeError:
        self.Vdot_array = np.array([self.Vdot])

    self.Pe_ij = np.zeros(
        [self.Vdot_array.size, self.T_array.size]
        )
    self.Da_j = np.zeros(self.T_array.size)
    self.eta_ij = np.zeros(self.Pe_ij.shape)

    for i in np.arange(self.Vdot_array.size):
        for j in np.arange(self.T_array.size):

            self.Vdot = self.Vdot_array[i]
            self.T = self.T_array[j]

            self.eta_ij[i, j] = self.get_eta(self.Vdot, self.T)

            self.Da_j[j] = self.Da
            self.Pe_ij[i, j] = self.Pe

def get_eta(self, *args, **kwargs):
```

```
        """Returns conversion efficiency.

        Inputs:

        Vdot : flow rate (m^3/s)
        T : temperature (K).

        or kwargs:
        Pe
        Da

        or if none:
        self.Vdot and self.T are used

        Vdot and T are generally going to be the independent variables
        that are varied here so they need to be inputs.
        """

        if 'Pe' in kwargs:
            Pe = kwargs['Pe']

        elif len(args) == 2:
            Vdot = args[0]
            T = args[1]
            Pe = self.get_Pe(Vdot, T)

        else:
            Vdot = self.Vdot
            T = self.T
            Pe = self.get_Pe(Vdot, T)

        if 'Da' in kwargs:
            Da = kwargs['Da']
            A_i = self.get_A_i(Da=Da)

        elif len(args) == 2:
            T = args[1]
            A_i = self.get_A_i(T)

        else:
            T = self.T
            A_i = self.get_A_i(T)

        self.eta_array = (
            A_i / self.lambda_i * np.sin(self.lambda_i) * (1. -
            np.exp(-4 * self.lambda_i ** 2. / Pe * self.x_))
            )

        self.eta = self.eta_array.sum()

        return self.eta

    def get_eta_fit(self, T_exp, A_arr, T_a):

        """Returns eta with inputs that are used by curve_fit

        Inputs:
        T: temperature (K)

        used by curve_fit as fit parameters:
        A_arr : pre-exponential coefficient for Arrhenius kinetics
        T_a : activation temperature (K)
```

```
        """

        self.A_arr = A_arr
        self.T_a = T_a

        self.T_array = T_exp
        self.set_eta_ij()

        self.eta_ij = self.eta_ij.reshape(self.eta_ij.size)

        return self.eta_ij

    def get_Y(self, x_, y_, **kwargs):

        """Sets non-dimensional Y at specified non-d (x, y) point.

        Inputs:

        x_ : streamwise coordinate scaled by channel height
        y_ : transverse coordinate scaled by channel height

        """

        # T and Vdot are generally going to be constants here so they
        # are not used as input arguments.

        if 'Pe' in kwargs:
            Pe = kwargs['Pe']

        else:
            T = self.T
            Vdot = self.Vdot
            Pe = self.get_Pe(Vdot, T)

        if 'Da' in kwargs:
            Da = kwargs['Da']
            A_i = self.get_A_i(Da=Da)
            lambda_i = self.lambda_i

        else:
            T = self.T
            A_i = self.get_A_i(T)
            lambda_i = self.lambda_i

        self.Y = (
            (A_i * np.exp(-4. * lambda_i ** 2. / Pe * x_) *
            np.cos(lambda_i * y_)).sum()
            )

        return self.Y

    def get_A_i(self, *args, **kwargs):

        """Returns pre-exponential Arrhenius coefficient.

        Inputs:

        T: temperature (K)

        or keyword argument Da from get_Da
        or keyword argument lambda_i from get_lambda
        """
```

```
    if len(args) == 1:
        T = args[0]
        lambda_i = self.get_lambda(T)

    elif 'Da' in kwargs:
        Da = kwargs['Da']
        lambda_i = self.get_lambda(Da=Da)

    elif 'lambda_i' in kwargs:
        lambda_i = kwargs['lambda_i']

    self.A_i = (
        2. * np.sin(lambda_i) / (lambda_i + np.sin(lambda_i) *
        np.cos(lambda_i))
        )

    return self.A_i

def get_lambda_spl(self, *args, **kwargs):

    """Uses spline fit to represent lambda as a function of Da.

    Inputs:

    T : temperature (K)

    or keyword argument Da from get_Da
    """

    if 'Da' in kwargs:
        Da = kwargs['Da']

    else:
        T = args[0]
        Da = np.float32(self.get_Da(T))

    self.lambda_i = np.zeros(self.terms)

    for i in range(self.terms):
        self.lambda_i[i] = (
            interp.splev(Da, self.lambda_splines[i])
            )

    return self.lambda_i

def get_lambda_error(self, guess, *args):

    """Returns error associated with guess of lambda.

    Inputs:
    guess: initial guess at lambda as a function of Da.

    if Da in kwargs, then Da is used
    """

    if len(args) == 1:
        Da = args[0]

    else:
        Da = self.Da

    error = 1 - guess / Da * np.tan(guess)
```

```
    return error

def get_lambda(self, *args, **kwargs):

    """Uses fsolve to represent lambda as a function of Da.

    Inputs:

    T : temperature (K)

    or keyword argument Da from get_Da
    """

    if 'Da' in kwargs:
        Da = kwargs['Da']

    else:
        T = args[0]
        Da = np.float32(self.get_Da(T))

    self.lambda_i = self.get_lambda_spl(Da=Da)
    lambda_i = self.lambda_i

    self.lambda_i = (
        fsolve(self.get_lambda_error, x0=lambda_i, args=(Da))
        )

    return self.lambda_i
```

Listing E.3: experimental.py

```
from scipy.optimize import curve_fit
import xlrd
import numpy as np

def get_S_r(self):

    """Returns sum of residuals squared for all data points."""

    S_r = np.sum((self.eta_model - self.eta_exp) ** 2.)

    return S_r

def import_data(self):

    """Imports data from excel sheet."""

    self.worksheet = (
        xlrd.open_workbook(filename=self.source).sheet_by_index(0)
        )
    # Import conversion data from worksheet and store as scipy arrays
    self.T_exp = np.array(
        self.worksheet.col_values(0, start_rowx=4, end_rowx=None)
        ) + 273.15
    self.HCout_raw = np.array(
        self.worksheet.col_values(4, start_rowx=4, end_rowx=None)
        )
    self.HCin_raw = np.array(
        self.worksheet.col_values(8, start_rowx=4, end_rowx=None)
        )
    self.eta_exp = (
        (self.HCin_raw - self.HCout_raw) / self.HCin_raw
        )
```

```
    self.T_model = np.linspace(
        self.T_exp[0] - 50, self.T_exp[-1] + 50, 25
        )
    self.T_array = self.T_model

def set_fit_params(self):

    """Uses scipy optimize curve_fit to determine Arrhenius
    parameters that result in best curve fit."""

    self.p0 = np.array([self.A_arr, self.T_a])
    # initial guess at A_arr and T_a

    self.popt, self.pcov = curve_fit(
        self.get_eta_fit, self.T_exp, self.eta_exp, p0=self.p0
        )

    self.A_arr = self.popt[0]
    self.T_a = self.popt[1]

    self.T_array = self.T_model
```

Listing E.4: numerical.py

```
import numpy as np
from scipy.integrate import odeint

def solve_numeric(self):

    """Solves for species and conversion numerically."""

    self.delta_x = self.x_array[1] - self.x_array[0]

    self.delta_y = self.y_array[1] - self.y_array[0]
    Y0 = np.ones(self.y_array.size)

    self.Yxy_num = odeint(self.get_Yprime, y0=Y0, t=self.x_array)

def set_eta_ij_num(self):

    """Sets conversion efficiency over a range of Pe and Da."""

    try:
        self.Vdot_array
    except AttributeError:
        self.Vdot_array = np.array([self.Vdot])

    self.Pe_ij = np.zeros(
        [self.Vdot_array.size, self.T_array.size]
        )
    self.Da_j = np.zeros(self.T_array.size)
    self.eta_ij_num = np.zeros(self.Pe_ij.shape)

    for i in np.arange(self.Vdot_array.size):
        for j in np.arange(self.T_array.size):

            self.Vdot = self.Vdot_array[i]
            self.T = self.T_array[j]

            self.eta_ij_num[i, j] = self.get_eta_num(self.Vdot, self.T)
            self.Da_j[j] = self.Da
            self.Pe_ij[i, j] = self.Pe
```

```
def get_eta_num(self, Vdot, T):

    """Returns conversion efficiency.

    Inputs:
    Vdot : flow rate (m^3/s)
    T : temperature (K).
    """

    self.Pe = self.get_Pe(Vdot, T)
    self.Da = self.get_Da(T)

    self.solve_numeric()

    x_array = self.x_array
    self.x_array = np.array([self.x_array[0], self.x_array[-1]])

    self.eta_num = (
        self.Yxy_num[0, :].mean() - self.Yxy_num[-1, :].mean()
        )

    self.x_array = x_array

    return self.eta_num

def get_Yprime(self, Y, x):

    """Returns Yprime for numerical solver."""

    Yprime = np.zeros(Y.size)

    # symmetry boundary condition
    Yprime[0] = (
        4. / self.Pe * (Y[1] - 2 * Y[0] + Y[1]) / self.delta_y ** 2
        )

    # in the channel
    for i in range(1, self.y_array.size - 1):
        Yprime[i] = (
            4. / self.Pe * (Y[i + 1] - 2 * Y[i] + Y[i - 1]) /
            self.delta_y ** 2
            )

    # wall BC
    Yprime[-1] = (
        8. / self.Pe * ((Y[-2] - Y[-1]) / self.delta_y - self.Da *
                        Y[-1]) / self.delta_y
        )

    return Yprime
```

Listing E.5: prop_functions.py

```
import numpy as np
import constants as const
reload(const)

def get_Da(self, *args):

    """Returns Damkoehler number.

    Inputs:
```

```
        T: temperature (K)

        of if none:
        self.T is used
        """

        if len(args) == 1:
            T = args[0]

        else:
            T = self.T

        thiele = self.get_thiele(T)

        self.Da = (
            0.5 * self.D_C3H8_air_eff / self.D_C3H8_air * self.height /
            self.thickness * np.sqrt(thiele) * np.tanh(np.sqrt(thiele))
           )

        return self.Da

    def get_thiele(self, T):

        """Returns Thiele modulus.

        Inputs:

        T: temperature (K)"""

        k_arr = (self.A_arr * np.exp(-self.T_a / T))
        D_C3H8_air_eff = self.get_D_C3H8_air_eff(T)

        self.thiele = (k_arr * self.thickness ** 2 / D_C3H8_air_eff)

        return self.thiele

    def get_k(self, T):

        """Returns Thiele modulus.

        Inputs:

        T: temperature (K)"""

        self.k_arr = self.A_arr * np.exp(-self.T_a / T)

        return self.k_arr

    def get_Pe(self, *args):

        """Returns Peclet number

        Inputs:

        Vdot : flow rate (m^3/s)
        T : temperature (K).

        if if none:
        Vdot = self.Vdot
        T = self.T

        """
```

```
        if len(args) == 2:
            Vdot = args[0]
            T = args[1]

        else:
            Vdot = self.Vdot
            T = self.T

        D_C3H8_air = self.get_D_C3H8_air(T)

        self.U = Vdot / (self.width * self.height) * (T / self.T_ambient)

        self.Pe = self.U * self.height / D_C3H8_air

        return self.Pe

    def set_TempPres_dependents(self, T):

        """Performance this function on both fuel and air.

        Input:
        T : temperature (K)

        Requires that self.P is set."""

        self.air.T = T
        self.air.P = self.P
        self.air.set_TempPres_dependents()
        self.fuel.T = T
        self.fuel.P = self.P
        self.fuel.set_TempPres_dependents()

    def get_mfp(self, T):

        """Returns crude approximation of mfp (m) of propane in air

        Method from Bird, Stewart, Lightfoot Eq. 17.3-3.

        Input:

        T : temperature (K)

        Output:

        mfp : mean free path (m) of air molecule"""

        self.air.T = T
        self.air.set_TempPres_dependents()

        self.mfp = (
            (np.sqrt(2.) * np.pi * self.air.d ** 2. * self.air.n) ** -1.
            )

        return self.mfp

    def get_Kn(self, T):

        """Returns Knudsen number for air.

        Inputs:

        T: temperature (K)
```

```
        self.Kn_length must be set.

        Returns
        -----------
        Kn : Knudsen number"""

        mfp = self.get_mfp(T)

        self.Kn = mfp / self.Kn_length

        return self.Kn

    def get_D_C3H8_air_eff(self, T):

        """Returns effective diffusion coefficient in porous media.

        Inputs:

        T: temperature (K)

        I need to put a refernce here for this scaling technique.  I'm
        pretty sure it is documented in the paper."""

        Kn = self.get_Kn(T)
        D_C3H8_air_Kn = self.get_D_C3H8_air_Kn(T)

        if np.isscalar(Kn):
            if Kn <= 1.:
                D_C3H8_air_eff = (
                    self.porosity / self.tortuosity * self.D_C3H8_air
                    )
            else:
                D_C3H8_air_eff = (
                    2. * self.porosity / self.tortuosity *
            (self.D_C3H8_air * D_C3H8_air_Kn) / (self.D_C3H8_air +
            D_C3H8_air_Kn)
                    )

        else:
            if Kn.any() <= 1.:
                D_C3H8_air_eff = (
                    self.porosity / self.tortuosity * self.D_C3H8_air
                    )
            else:
                D_C3H8_air_eff = (
                    2. * self.porosity / self.tortuosity *
            (self.D_C3H8_air * D_C3H8_air_Kn) / (self.D_C3H8_air +
            D_C3H8_air_Kn)
                    )

        self.D_C3H8_air_eff = D_C3H8_air_eff

        return D_C3H8_air_eff

    def get_D_C3H8_air_Kn(self, T):

        """Returns Knudsen diffusion coefficient for fuel/air.

        T: temperature (K)
        """

        Kn = self.get_Kn(T)
        D_C3H8_air = self.get_D_C3H8_air(T)
```

```
        self.D_C3H8_air_Kn = D_C3H8_air / Kn

        return self.D_C3H8_air_Kn

    def get_D_C3H8_air(self, T):

        """Returns binary diffusion coefficient for fuel/air.

        Method is from Bird, Stewart, Lightfoot Transport Phenomena
        2nd Ed. Equation 17.3-10

        Inputs:
        T : temperature

        Output:

        mfp : mean free path (m) of air molecule"""

        self.set_TempPres_dependents(T)

        self.D_C3H8_air = (
            2. / 3. * np.sqrt(const.k_B * T / np.pi * 0.5 * (1. /
            self.air.m + 1. / self.fuel.m)) / (np.pi * (0.5 *
            (self.air.d + self.fuel.d)) ** 2.) / self.air.n
            )

        return self.D_C3H8_air
```

E.2 Waste Heat Recovery Code

All code in this section can be found here:

https://github.com/calbaker/TE_Model.

Listing E.6: hx.py

```
# coding=utf-8
"""
Script defining HX class.

Chad Baker
Created on 2011 Feb 10
"""

# Distribution Modules
import time
import numpy as np
import operator
from scipy.optimize import fmin  # _l_bfgs_b

# User Defined Modules
# In this directory
import engine
import te_pair
reload(te_pair)
import exhaust
reload(exhaust)
```

```
import coolant
reload(coolant)
import platewall
reload(platewall)


class Dimension(object):
    """Class for hx attribute containing physical dimensions.  This is
    used on an ad hoc basis."""
    pass


class HX(object):

    """Class definition for heat exchanger.

    Class instances:

    cool : coolant.Coolant instance
    cummins : engine.Engine instance
    exh : exhaust.Exhaust instance
    plate : platewall.PlateWall instance
    te_pair : te_pair.TE_pair instance

    Methods:

    fix_geometry
    get_T_inlet_error
    get_minpar
    init_arrays
    optimize
    set_availability
    set_constants
    set_convection
    set_mdot_charge
    setup
    solve_hx
    solve_node
    store_node_values
    """

    def __init__(self):

        """Sets several attributes, including instance attributes.

        Instance attributes

        self.cool = coolant.Coolant()
        self.exh = exhaust.Exhaust()
        self.te_pair = te_pair.TE_Pair()
        self.plate = platewall.PlateWall()
        self.cummins = engine.Engine()

        Methods:

        self.fix_geometry

        """

        self.R_extra = 0.
        # Dummy variable that can be used as a fit parameter or for
        # any other appropriate purpose to add thermal resistance to
        # the model
```

```
self.R_interconnect = 0.00075  # (m^2*K/kW)
# Resistance of copper interconnect assuming a thickness of
# 0.3 mm (Ref: Hori, Y., D. Kusano, T. Ito, and
# K. Izumi.  Analysis on Thermo-mechanical Stress of
# Thermoelectric Module.  In Thermoelectrics 1999. Eighteenth
# International Conference On, 328  331, 1999), where
# k_interconnect = 400 W/(m-K)

self.R_substrate = 0.005  # (m^2*K/kW)
# resistance of ceramic substrate(AlN) 1 mmm thick (Hori, Y.,
# D. Kusano, T. Ito, and K. Izumi.  Analysis on
# Thermo-mechanical Stress of Thermoelectric Module.  In
# Thermoelectrics, 1999. Eighteenth International Conference
# On, 328  331, 1999.), based on k_ceramic = 200 W/(m-K)
# obtained from Thermoelectrics Handbook.

self.R_contact = 0.00003  # (m^2*K/kW)
# Thermal contact resistance for all three contacts estimated
# using alumina/copper contact resistance extracted from
# Gundrum, Bryan C., David G. Cahill, and Robert
# S. Averback.  Thermal Conductance of Metal-metal
# Interfaces.  Physical Review B 72, no. 24 (December 30,
# 2005): 245426.

# self.R_contact = 0.8322 # (m^2*K/kW)
# thermal contact resistance (m^2*K/kW) for plate/substrate,
# substrate/interconnect, and interconnect/TE leg interfaces
# all combined. All estimated (at 450 K) based on AlN/Cu
# contact resistance extracted from Shi, Ling, Gang Wu,
# Hui-ling Wang, and Xin-ming Yu.  Interfacial Thermal Contact
# Resistance Between Aluminum Nitride and Copper at Cryogenic
# Temperature.  Heat and Mass Transfer 48, no. 6 (2012):
# 999  1004.

self.dimension = Dimension()
self.dimension.width = 0.55
# width (cm*10**-2) of HX duct. This model treats duct as
# parallel plates for simpler modeling.
self.dimension.length = 0.55
# length (m) of HX duct
self.nodes = 25
# number of nodes for numerical heat transfer model
self.x0 = np.array([.7, 0.02, 0.001, 4.])
self.xb = [(0.5, 2.), (0., 1.), (1.e-4, 20.e-3), (0.1, None)]
# initial guess and bounds for x where entries are N/P area,
# fill fraction, leg length (m), and current (A)
self.xmin_file = 'xmin'
self.T0 = 300.
# temperature (K) at restricted dead state
self.equal_width = True

self.apar_list = [
    ['self', 'te_pair', 'leg_area_ratio'],
    ['self', 'te_pair', 'fill_fraction'],
    ['self', 'te_pair', 'length'],
    ['self', 'te_pair', 'I']
    ]
# list of strings used to construct names of attributes to be
# optimized

# initialization of instance attributes
self.cool = coolant.Coolant()
```

```
        self.exh = exhaust.Exhaust()
        self.te_pair = te_pair.TE_Pair()
        self.plate = platewall.PlateWall()
        self.cummins = engine.Engine()

        self.arrangement = 'single'
        self.fix_geometry()

    def init_arrays(self):

        """Initializes arrays for storing node values."""

        self.Qdot_nodes = np.zeros(self.nodes)

        self.exh.Vdot_nodes = np.zeros(self.nodes)

        self.exh.T_nodes = np.zeros(self.nodes)
        self.exh.h_nodes = np.zeros(self.nodes)
        self.exh.f_nodes = np.zeros(self.nodes)
        self.exh.deltaP_nodes = np.zeros(self.nodes)
        self.exh.Wdot_nodes = np.zeros(self.nodes)
        self.exh.Nu_nodes = np.zeros(self.nodes)
        self.exh.C_nodes = np.zeros(self.nodes)
        self.exh.c_p_nodes = np.zeros(self.nodes)
        self.exh.mu_nodes = np.zeros(self.nodes)
        self.exh.entropy_nodes = np.zeros(self.nodes)
        self.exh.enthalpy_nodes = np.zeros(self.nodes)
        self.exh.velocity_nodes = np.zeros(self.nodes)
        self.exh.rho_nodes = np.zeros(self.nodes)
        self.exh.Re_nodes = np.zeros(self.nodes)

        self.cool.T_nodes = np.zeros(self.nodes)
        self.cool.entropy_nodes = np.zeros(self.nodes)
        self.cool.enthalpy_nodes = np.zeros(self.nodes)
        self.cool.deltaP_nodes = np.zeros(self.nodes)
        self.cool.Wdot_nodes = np.zeros(self.nodes)

        self.U_hot_nodes = np.zeros(self.nodes)
        self.U_cold_nodes = np.zeros(self.nodes)

        self.te_pair.q_h_conv_nodes = np.zeros(self.nodes)
        self.te_pair.q_c_conv_nodes = np.zeros(self.nodes)
        self.te_pair.q_h_nodes = np.zeros(self.nodes)
        self.te_pair.q_c_nodes = np.zeros(self.nodes)
        self.te_pair.error_nodes = np.zeros([3, self.nodes])
        self.te_pair.T_c_nodes = np.zeros(self.nodes)
        self.te_pair.T_h_nodes = np.zeros(self.nodes)
        self.te_pair.h_nodes = np.zeros(self.nodes)
        self.te_pair.power_nodes = np.zeros(self.nodes)
        self.te_pair.power_nodes_check = np.zeros(self.nodes)
        self.te_pair.eta_nodes = np.zeros(self.nodes)

    def setup(self):

        """Sets attributes that must be defined before running model.

        Methods:

        self.set_mdot_charge
        self.set_constants

        Useful for terminal.  Not necessary elsewhere.
```

```
        """

        self.exh.T = 800.
        self.cool.T = 300.
        self.set_mdot_charge()
        self.set_constants()

    def set_constants(self):

        """Sets constants used at the HX level.

        Methods:

        self.fix_geometry
        self.exh.set_flow_geometry
        self.cool.set_flow_geometry

        """

        self.x = np.linspace(0, self.dimension.length, self.nodes)
        self.node_length = self.dimension.length / self.nodes
        # length (m) of each node
        self.area = self.node_length * self.dimension.width * self.cool.ducts
        # area (m^2) through which heat flux occurs in each node
        self.te_pair.set_constants()
        self.leg_pairs = self.area / self.te_pair.area
        # Number of TEM leg pairs per node
        self.x_dim = np.arange(self.node_length / 2, self.dimension.length +
        self.node_length / 2, self.node_length)
        # x coordinate (m)
        self.fix_geometry()
        self.exh.set_flow_geometry(self.exh.width)
        self.cool.set_flow_geometry(self.cool.width)

    def fix_geometry(self):

        """Matches geometry of ducts.

        Makes sure that common geometry like width and length is the
        same between exh, cool, and the overal heat exchanger.

        """

        if self.equal_width == True:
            self.exh.width = self.dimension.width
            self.cool.width = self.dimension.width
        self.cool.length = self.dimension.length
        self.exh.length = self.dimension.length

    def set_mdot_charge(self):

        """Sets exhaust mass flow rate.

        Methods:

        self.cummins.set_mdot_charge

        Eventually, this should be a function of speed, load, and EGR
        fraction.  Also, it should come from experimental data.  Also,
        it should probably go within the exhaust module.

        """
```

```
        self.cummins.set_mdot_charge()
        # mass flow rate (kg/s) of exhaust
        self.exh.mdot = self.cummins.mdot_charge

    def set_convection(self):

        """Sets values for convection coefficients.

        Methods:

        self.exh.set_flow
        self.cool.set_flow

        """

        # Exhaust stuff
        self.exh.set_flow()
        # Coolant stuff
        self.cool.set_flow()
        # The previous three commands need only execute once per node.

        self.U_hot = ((self.exh.R_thermal + self.R_parasitic) ** -1)
        # heat transfer coefficient (kW/m^-K) between TE hot side and
        # exhaust
        self.U_cold = ((self.cool.R_thermal + self.R_parasitic) ** -1)
        # heat transfer coefficient (kW/m^-K) between TE cold side and
        # coolant

    def solve_node(self, i):

        """Solves for performance of streamwise slice of HX.

        Methods:

        self.set_convection
        self.te_pair.solve_te_pair

        """

        self.te_pair.T_h_conv = self.exh.T
        self.te_pair.T_c_conv = self.cool.T

        self.set_convection()

        if i == 0:
            self.te_pair.T_h = self.exh.T
            # guess at hot side TEM temperature (K)
            self.te_pair.T_c = self.cool.T
            # guess at cold side tem temperature (K)

        self.te_pair.U_hot = self.U_hot
        self.te_pair.U_cold = self.U_cold

        self.te_pair.solve_te_pair()
        self.q_h = self.te_pair.q_h
        self.q_c = self.te_pair.q_c

        self.Qdot_node = self.q_h * self.area
        # heat transfer on hot side of node, positive values indicates
        # heat transfer from hot to cold

    def solve_hx(self, ** kwargs):
        # solve parallel flow heat exchanger
```

```
"""Solves for performance of all stream-wise nodes.

Methods:

self.init_arrays
self.set_constants
self.solve_node
self.store_node_values
self.set_availability

"""

self.init_arrays()
self.set_constants()

self.R_parasitic = (self.plate.R_thermal + self.R_interconnect +
self.R_substrate + self.R_contact + self.R_extra)
# R_parasitic (m^2-K/kW) includes plate resistance from module
# platewall, resistance of interconnect and ceramic substrate
# and all the contact resistances

self.exh.node_length = self.node_length
self.exh.T = self.exh.T_inlet
# T_inlet and T_outlet correspond to the temperatures going
# into and out of the heat exchanger.
if self.type == 'parallel':
    self.cool.T = self.cool.T_inlet
elif self.type == 'counter':
    self.cool.T = self.cool.T_outlet
self.cool.node_length = self.node_length

# for loop iterates of nodes of HX in streamwise direction
for i in np.arange(self.nodes):
    self.solve_node(i)
    self.store_node_values(i)

    # redefining temperatures (K) for next node
    self.exh.T = (self.exh.T - self.te_pair.q_h * self.area /
        self.exh.C)
    if self.type == 'parallel':
        self.cool.T = (self.cool.T + self.te_pair.q_c * self.area
            / self.cool.C)
    elif self.type == 'counter':
        self.cool.T = (self.cool.T - self.te_pair.q_c * self.area
            / self.cool.C)

# defining HX outlet/inlet temperatures (K)
self.exh.T_outlet = self.exh.T
if self.type == 'parallel':
    self.cool.T_outlet = self.cool.T
elif self.type == 'counter':
    self.cool.T_inlet = self.cool.T

self.Qdot_total = self.Qdot_nodes.sum()
self.Qdot_max = (
    self.exh.C_nodes.mean() * (self.exh.T_inlet -
self.cool.T_inlet)
    )
self.exh.Re_omega = (
    self.exh.D / (self.exh.flow_area *
self.exh.mu_nodes.mean()) * self.exh.mdot_omega
    )
```

```
        self.effectiveness = (
            self.Qdot_total / self.Qdot_max
            )
        # heat exchanger effectiveness

        self.te_pair.power_total = self.te_pair.power_nodes.sum()
        # total TE power output (kW)

        self.exh.deltaP_total = self.exh.deltaP_nodes.sum()
        self.cool.deltaP_total = self.cool.deltaP_nodes.sum()

        self.exh.Wdot_total = self.exh.Wdot_nodes.sum()
        self.cool.Wdot_total = self.cool.Wdot_nodes.sum()

        # patch to handle the minor losses for the IdealFin
        # enhancement
        try:
            self.exh.enh.type
        except AttributeError:
            pass
        else:
            if self.exh.enh.type == 'IdealFin':
                self.exh.K_c = 0.4
                self.exh.K_e = 0.2

                self.exh.deltaP_minor_in = (
                    (0.5 * self.exh.rho_nodes[0] *
                    self.exh.velocity_nodes[0] ** 2. * (self.exh.K_c
                    + 1. - self.exh.sigma ** 2.) * 1.e-3)
                    )
                self.exh.deltaP_minor_out = (
                    -(0.5 *  self.exh.rho_nodes[-1] *
                    self.exh.velocity_nodes[-1] ** 2. * (1. - self.exh.sigma
                    ** 2. - self.exh.K_e) *
                    self.exh.velocity_nodes[-1] /
                    self.exh.velocity_nodes[0]) * 1.e-3
                    )

                self.exh.deltaP_minor = (
                    self.exh.deltaP_minor_in +
                    self.exh.deltaP_minor_out
                    )
                self.exh.deltaP_total += self.exh.deltaP_minor
                self.exh.Wdot_minor = (
                    self.exh.Vdot_nodes[0] * self.exh.deltaP_minor_in
                    + self.exh.Vdot_nodes[-1] *
                    self.exh.deltaP_minor_out
                    )
                self.exh.Wdot_total += self.exh.Wdot_minor

        self.Wdot_pumping = (self.exh.Wdot_total +
                             self.cool.Wdot_total)
        # total pumping power requirement (kW)

        self.power_net = (
            self.te_pair.power_total - self.Wdot_pumping
            )

        self.set_availability()

    def set_availability(self):

        """Sets availability of exhaust and coolant along all nodes.
```

```
        """

        # Availability analysis
        self.exh.enthalpy0 = self.exh.get_enthalpy(self.T0)
        # enthalpy (kJ/kg) of exhaust at restricted dead state
        self.exh.entropy0 = self.exh.get_entropy(self.T0)
        # entropy (kJ/kg*K) of exhuast at restricted dead state

        self.exh.availability_flow_nodes = ((self.exh.enthalpy_nodes
        - self.exh.enthalpy0 - self.T0 * (self.exh.entropy_nodes -
        self.exh.entropy0)) * self.exh.mdot)
        # availability (kJ/kg) of exhaust

        self.cool.enthalpy_nodes = (self.cool.c_p *
        (self.cool.T_nodes - self.T0) + self.cool.enthalpy0)
        # enthalpy (kJ/kg*K) of coolant
        self.cool.entropy_nodes = (self.cool.c_p *
        np.log(self.cool.T_nodes / self.T0) + self.cool.entropy0)

        self.cool.availability_flow_nodes = (
        (self.cool.enthalpy_nodes - self.cool.enthalpy0 - self.T0 *
        (self.cool.entropy_nodes - self.cool.entropy0)) *
        self.cool.mdot)
        # availability (kJ/kg) of coolant

    def store_node_values(self, i):

        """Stores values of parameters of interest in node i.

        This should eventually also store the node valuves for T, q,
        and material properties in the te legs.
        """

        self.Qdot_nodes[i] = self.Qdot_node
        # storing node hot side heat transfer in array

        self.te_pair.q_h_conv_nodes[i] = self.q_h
        self.te_pair.q_c_conv_nodes[i] = self.q_c
        self.te_pair.q_h_nodes[i] = self.te_pair.q_h
        self.te_pair.q_c_nodes[i] = self.te_pair.q_c
        self.te_pair.error_nodes[:, i] = self.te_pair.error
        self.te_pair.T_h_nodes[i] = self.te_pair.T_h
        self.te_pair.T_c_nodes[i] = self.te_pair.T_c
        self.te_pair.power_nodes[i] = self.te_pair.P * self.leg_pairs
        self.te_pair.power_nodes_check[i] = (
            self.te_pair.P_flux * self.area
            )
        self.te_pair.eta_nodes[i] = self.te_pair.eta
        self.te_pair.h_nodes[i] = self.te_pair.h_eff

        self.exh.T_nodes[i] = self.exh.T
        self.exh.Vdot_nodes[i] = self.exh.Vdot
        self.exh.f_nodes[i] = self.exh.f
        self.exh.deltaP_nodes[i] = self.exh.deltaP
        self.exh.Wdot_nodes[i] = self.exh.Wdot_pumping
        self.exh.Nu_nodes[i] = self.exh.Nu_D
        self.exh.C_nodes[i] = self.exh.C
        self.exh.c_p_nodes[i] = self.exh.c_p
        self.exh.mu_nodes[i] = self.exh.mu
        self.exh.h_nodes[i] = self.exh.h_conv
        self.exh.velocity_nodes[i] = self.exh.velocity
        self.exh.entropy_nodes[i] = self.exh.entropy
        self.exh.enthalpy_nodes[i] = self.exh.enthalpy
```

```
        self.exh.rho_nodes[i] = self.exh.rho
        self.exh.Re_nodes[i] = self.exh.Re_D

        self.cool.T_nodes[i] = self.cool.T
        self.cool.deltaP_nodes[i] = self.cool.deltaP
        self.cool.Wdot_nodes[i] = self.cool.Wdot_pumping

        self.U_hot_nodes[i] = self.U_hot
        self.U_cold_nodes[i] = self.U_cold

    def get_minpar(self, apar):

        """Returns inverse of net power.

        Methods:

        self.solve_hx
        self.set_leg_areas

        Used by method self.optimize

        Uses self.apar_list to determine which paramters are to be
        varied in optimization.  Use with scipy.optimize.fmin to find
        optimal set of input parameters."""

        self.opt_iter = self.opt_iter + 1
        if self.opt_iter % 15 == 0:
            print "\n\noptimizaton iteration", self.opt_iter
            print "net power", self.power_net
            for i in range(self.x0.size):
                varname = '.'.join(self.apar_list[i][1:])
                varval = (
                    operator.attrgetter(varname)(self)
                    )
                print varname + ":", varval

            print "leg pairs =", self.leg_pairs

        apar = np.array(apar)

        # unpack guess vector
        for i in range(apar.size):
            setattr(operator.attrgetter(
                    '.'.join(self.apar_list[i][1:-1]))(self),
            self.apar_list[i][-1], apar[i])

        # reset surrogate variables
        self.te_pair.set_leg_areas()

        self.solve_hx()

        if (apar <= 0.).any():
            minpar = np.abs(self.power_net) ** 3 + 100.
            # penalizes negative parameters
            print "Encountered impossible value."

        else:
            minpar = - self.power_net

        return minpar

    def optimize(self):
```

```
        """Finds optimal set of paramters in self.apar_list

        Methods:

        self.get_minpar

        self.x0 and self.xb must be defined elsewhere."""

        time.clock()

        # dummy function that might be used with minimization
        def fprime():
            return 1

        self.opt_iter = 0

        self.x0 = np.zeros(len(self.apar_list))

        for i in range(self.x0.size):
            self.x0[i] = (
            operator.attrgetter('.'.join(self.apar_list[i][1:]))(self)
           )

        self.xmin = fmin(self.get_minpar, self.x0)

        t1 = time.clock()

        print '\n'
        for i in range(self.x0.size):
            varname = '.'.join(self.apar_list[i][1:])
            varval = (
                operator.attrgetter(varname)(self)
                )
            print varname + ":", varval

        print "\npower net:", self.power_net * 1000., 'W'
        print "power raw:", self.te_pair.power_total * 1000., 'W'
        print "pumping power:", self.Wdot_pumping * 1000., 'W'
        self.exh.volume = (
            self.exh.height * self.exh.width * self.dimension.length
            )
        print "exhaust volume:", self.exh.volume * 1000., 'L'
        VAR = self.power_net / self.exh.volume
        print "exhaust power density:", VAR, 'kW/m^3'

        print """Elapsed time solving xmin1 =""", t1

    def save_opt_par(self, opt_par_dir):
        """Saves parameters found by optimize."""

        if self.opt_iter == 0:
            print """\nError. Script must run the function optimize
            before running save_opt_par."""

        for i in range(self.x0.size):
            varname = '.'.join(self.apar_list[i][1:])
            varval = (
                operator.attrgetter(varname)(self)
                )
            np.save(opt_par_dir + varname, varval)

    def get_T_inlet_error(self, T_outlet):
```

```
        """Returns error for coolant inlet temperature.

        Error is determined relative to desired setpoint inlet
        temperature for the counter flow configuration in which the
        outlet coolant temperaure is specified.  Should be used with
        fsolve to determine the correct inlet temperature for the
        coolant.

        Inputs:

        self.cool.T_outlet

        Methods:

        self.solve_hx

        """

        self.cool.T_outlet = np.float(T_outlet)
        self.solve_hx()
        error = self.cool.T_inlet_set - self.cool.T_inlet
        return error
```

Listing E.7: te_pair.py

```
# Distribution modules

import numpy as np
import time
from scipy.optimize import fsolve

# User defined modules
import leg
reload(leg)


class TE_Pair(object):
    """Class definition for TE leg pair with convection BC

    Methods:

    __init__
    get_error
    set_A_opt
    set_TEproperties
    set_ZT
    set_area
    set_constants
    set_eta_max
    set_power_max
    set_q_c_guess
    solve_te_pair
    solve_te_pair_once

    """

    def __init__(self):

        """Sets attributes and instantiates classes.

        Class instances:

        self.Ptype = leg.Leg()
```

```
        self.Ntype = leg.Leg()

        Methods:

        self.set_constants

        """

        self.leg_area_ratio = 0.7
        # Ratio of cross-section area of N-type leg to cross-section
        # area of P-type leg
        self.fill_fraction = 0.03
        # Percentage of nominal area occupied by TE legs.  This is not
        # consistent with the value for the area_void, unless or until
        # set_leg_areas has been called.
        self.length = 1.e-3
        # Length (m) of TE legs
        self.I = 1.  # electrical current (Amps)
        self.Ptype = leg.Leg()
        self.Ntype = leg.Leg()
        self.Ptype.material = 'HMS'
        self.Ntype.material = 'MgSi'
        self.area_void = (1.e-3) ** 2
        # Void area (m^2) associated with each leg pair.  This will be
        # changed when set_leg_areas is called.  After this happens,
        # it will be consisent with self.fill_fraction.
        self.length = 1.e-3
        self.nodes = 10
        #  number of nodes for which the temperature values are
        #  returned by odeint.  This does not affect the actual
        #  calculation, only the values for which results are stored.

        self.set_constants()

    def set_constants(self):

        """Sets a bunch of attributes that are usually held constant.

        Methods:

        self.Ntype.set_constants
        self.Ptype.set_constants
        self.set_leg_areas

        """

        self.Ntype.length = self.length
        self.Ptype.length = self.length
        self.Ptype.nodes = self.nodes
        self.Ntype.nodes = self.nodes
        self.Ptype.I = self.I
        # Current must have same sign as heat flux for p-type
        # material. Heat flux is negative because temperature gradient
        # is positive.
        self.Ntype.I = - self.I

        self.set_leg_areas()
        self.Ntype.set_constants()
        self.Ptype.set_constants()

    def solve_te_pair_once(self):

        """Solves legs and combines results of leg pair.
```

```
        Methods:

        self.Ntype.solve_leg_once
        self.Ptype.solve_leg_once

        """

        self.Ntype.solve_leg_once(self.Ntype.q_h)
        self.Ptype.solve_leg_once(self.Ptype.q_h)
        self.T_c = self.Ntype.T_c

        self.q_h = (
            (self.Ptype.q_h * self.Ptype.area + self.Ntype.q_h *
             self.Ntype.area) / self.area * 0.001
            )
        # area averaged hot side heat flux (kW/m^2)
        self.q_c = (
            (self.Ptype.q_c * self.Ptype.area + self.Ntype.q_c *
             self.Ntype.area) / self.area * 0.001
            )
        # area averaged hot side heat flux (kW/m^2)

        self.h_eff= self.q_h / (self.T_h - self.T_c)
        # effective coeffient of convection (kW/m^2-K)
        self.R_thermal = 1. / self.h_eff

    def get_error(self, knob_arr):

        """Returns BC error.

        This function uses guesses the hot side temperature and
        heat fluxes for both legs to solve the pair a single time.
        The resulting errors in boundary conditions are then
        determined. This is then used by fsolve in solve_te_pair to
        zero out the error between hot and cold side heat fluxes and
        the error between the cold side temperatures of both n-type
        and p-type devices.

        Methods:

        self.solve_te_pair_once

        """

        self.Ntype.q_h = knob_arr[0]
        self.Ptype.q_h = knob_arr[1]
        self.T_h = knob_arr[2]

        self.Ptype.T_h = self.T_h
        self.Ntype.T_h = self.T_h

        self.solve_te_pair_once()

        self.q_c_conv = self.U_cold * (self.T_c - self.T_c_conv)
        self.q_h_conv = self.U_hot * (self.T_h_conv - self.T_h)

        T_c_error = self.Ntype.T_c - self.Ptype.T_c
        q_c_error = self.q_c - self.q_c_conv
        q_h_error = self.q_h - self.q_h_conv

        self.error = np.array([T_c_error, q_c_error, q_h_error]).flatten()
```

```
        return self.error

    def set_q_guess(self):

        """Sets cold side guess for both Ntype and Ptype legs.

        Methods:

        self.Ntype.set_q_guess
        self.Ptype.set_q_guess

        """

        self.Ntype.set_q_guess()
        self.Ptype.set_q_guess()

    def solve_te_pair(self):

        """Solves legs and combines results of leg pair.

        Methods:

        self.set_q_guess

        """

        self.Ptype.T_h = self.T_h_conv
        self.Ntype.T_h = self.T_h_conv
        self.Ptype.T_c = self.T_c_conv
        self.Ntype.T_c = self.T_c_conv

        self.set_q_guess()
        knob_arr0 = np.array([self.Ntype.q_h_guess,
        self.Ptype.q_h_guess, self.T_h_conv])

        self.Ptype.T_c_goal = None
        self.Ntype.T_c_goal = None

        self.fsolve_output = fsolve(self.get_error, x0=knob_arr0)

        self.P = (self.Ntype.P + self.Ptype.P) * 0.001
        # power for the entire leg pair(kW). Negative sign makes this
        # a positive number. Heat flux is negative so efficiency needs
        # a negative sign also.
        self.P_flux = self.P / self.area
        # power flux (kW / m^2) through leg pair
        self.eta = self.P / (self.q_h * self.area)
        self.Vs = -self.Ntype.Vs + self.Ptype.Vs
        self.V = -self.Ntype.V + self.Ptype.V
        self.R_load = self.Ntype.R_load + self.Ptype.R_load
        self.R_internal = ( self.Ntype.R_internal +
        self.Ptype.R_internal )

    def set_TEproperties(self, T_props):

        """Sets properties for both legs based on temperature.

        Methods:

        self.Ntype.set_TEproperties(T_props)
        self.Ptype.set_TEproperties(T_props)

        """
```

```
        self.Ntype.set_TEproperties(T_props)
        self.Ptype.set_TEproperties(T_props)

    def set_ZT(self):

        """Sets ZT based on whatever properties were used last."""

        self.ZT = ( ((self.Ptype.alpha - self.Ntype.alpha) /
        ((self.Ptype.rho * self.Ptype.k) ** 0.5 + (self.Ntype.rho *
        self.Ntype.k) ** 0.5)) ** 2. * self.T_props )

    def set_eta_max(self):

        """Sets theoretical maximum efficiency.

        Methods:

        self.set_TEproperties(T_props)

        Uses material properties evaluated at the average temperature
        based on Sherman's analysis.

        """

        self.T_props = 0.5 * (self.T_h + self.T_c)
        self.set_TEproperties(T_props=self.T_props)
        self.set_ZT()
        delta_T = self.T_h - self.T_c
        self.eta_max = ( delta_T / self.T_h * ((1. + self.ZT) ** 0.5 -
        1.) / ((1. + self.ZT) ** 0.5 + self.T_c / self.T_h) )

    def set_A_opt(self):

        """Sets Ntype / Ptype area that results in max efficiency.

        Methods:

        self.set_TEproperties(T_props)

        Based on material properties evaluated at the average
        temperature.

        """

        self.set_TEproperties(T_props=self.T_props)
        self.A_opt = np.sqrt(self.Ntype.rho * self.Ptype.k /
        (self.Ptype.rho * self.Ntype.k))

    def set_power_max(self):

        """Sets power factor and maximum theoretical power.

        Methods:

        self.Ntype.set_power_factor
        self.Ptype.set_power_factor

        """

        self.Ntype.set_power_factor()
        self.Ptype.set_power_factor()
        self.power_max = self.Ntype.power_max + self.Ptype.power_max
```

```
def set_leg_areas(self):

    """Sets leg areas and void area.

    Based on leg area ratio and fill fraction.

    self.Ptype.area must be held constant.  self.Ntype.area and
    self.area_void are varied here.

    """

    leg_area_ratio = self.leg_area_ratio
    fill_fraction = self.fill_fraction

    self.Ntype.area = self.Ptype.area * leg_area_ratio
    self.area_void = (
        (1. - fill_fraction) / fill_fraction * (self.Ptype.area +
    self.Ntype.area)
        )
    self.area = self.Ntype.area + self.Ptype.area + self.area_void

def get_minpar(self, apar):

    """Returns inverse of power flux.

    Methods:

    self.set_leg_areas

    Used by method self.optimize

    self.length = apar[0]
    self.fill_fraction = apar[1]
    self.I = apar[2]
    self.leg_area_ratio = apar[3]

    Use with scipy.optimize.fmin to find optimal set of input
    parameters.

    This method uses power flux rather than power because for
    optimal power, leg height approaches zero and void area
    approaches infinity.  This trivial result is not useful."""

    self.opt_iter = self.opt_iter + 1
    if self.opt_iter % 15 == 0:
        print "\noptimizaton iteration", self.opt_iter
        print "leg length =", self.length, "m"
        print "fill fraction =", self.fill_fraction * 100., "%"
        print "current =", self.I, "A"
        print "area ratio =", self.leg_area_ratio
        print "power flux (kW/m^2)", self.P_flux
    apar = np.array(apar)

    self.length = apar[0]
    self.fill_fraction = apar[1]
    self.I = apar[2]
    self.leg_area_ratio = apar[3]

    # reset surrogate variables
    self.set_constants()

    self.solve_te_pair()
```

```
        if (apar <= 0.).any():
            minpar = np.abs(self.P_flux) ** 3. + 100
            print "Encountered impossible value."

        else:
            minpar = - self.P_flux

        return minpar

    def optimize(self):

        """Minimizes self.get_minpar

        Methods:

        self.get_minpar

        self.x0 and self.xb must be defined elsewhere."""

        time.clock()

        # dummy function that might be used with minimization
        def fprime():
            return 1

        self.opt_iter = 0

        self.x0 = np.array([self.length, self.fill_fraction,
        self.I, self.leg_area_ratio])

        from scipy.optimize import fmin

        self.xmin = fmin(self.get_minpar, self.x0)

        t1 = time.clock()

        print '\n'

        print "Optimized parameters:"
        print "leg length =", self.length, "m"
        print "fill fraction =", self.fill_fraction * 100., "%"
        print "current =", self.I, "A"
        print "area ratio =", self.leg_area_ratio

        print "\npower:", self.P * 1000., 'W'
        print "power flux:", self.P_flux, "kW/m^2"

        print """Elapsed time solving xmin1 =""", t1
```

Listing E.8: leg.py

```
# Distribution modules

import types
import numpy as np
from scipy.integrate import odeint
from numpy.testing import assert_approx_equal
from scipy.optimize import fsolve

# User defined modules
import mat_prop
reload(mat_prop)
```

```
class Leg(object):

    """Class for individual TE leg.

    Methods:

    __init__
    get_dTq_dx
    set_ZT
    set_constants
    set_power_factor
    set_q_guess
    solve_leg_anal
    solve_leg_once
    solve_leg
    get_error

    """

    def __init__(self):

        """Sets constants and binds methods.

        Methods:

        self.set_constants

        Binds the following methods:

        mat_prop.import_raw_property_data
        mat_prop.set_properties_v_temp
        mat_prop.set_TEproperties"""

        self.I = 0.5  # current (A) in TE leg pair
        self.nodes = 10
        # number of nodes for which values are stored
        self.length = 1.e-3  # leg length (m)
        self.area = (3.e-3) ** 2.  # leg area (m^2)

        self.C = 1.e7
        # assumed value for heat capacity (kJ / K)
        self.t_array = np.linspace(0, 5, 10)
        # array of times for transient solution

        self.set_constants()

        self.import_raw_property_data = (
            types.MethodType(mat_prop.import_raw_property_data, self)
            )
        self.set_properties_v_temp = (
            types.MethodType(mat_prop.set_properties_v_temp, self)
            )
        self.set_TEproperties = (
            types.MethodType(mat_prop.set_TEproperties, self)
            )

    def set_constants(self):

        """Sets attributes that are typically held constant."""

        self.x = np.linspace(0., self.length, self.nodes)
```

```
        self.J = self.I / self.area  # (Amps/m^2)

    def get_dTq_dx(self, Tq, x):

        """Solves node. Returns array of derivatives.

        Function for evaluating the derivatives of temperature and
        heat flux w.r.t. x-dimension

        Inputs:

        Tq : initial conditions
        x : array of locations where results are desired

        Methods:

        self.set_TEproperties(T_props)

        If there were a function called solve_node, it would do the
        same thing as this.

        """

        T = Tq[0]
        q = Tq[1]

        self.set_TEproperties(T)

        dT_dx = (
            1. / self.k * (self.J * T * self.alpha - q)
            )

        self.set_ZT()

        dq_dx = (
            (self.rho * self.J ** 2. * (1. + self.ZT)) - self.J *
            self.alpha * q / self.k
            )

        dVs_dx = self.alpha * dT_dx
        # Seebeck voltage, aka open circuit voltage

        dV_dx = self.alpha * dT_dx + self.rho * self.J
        # Seebeck voltage minus resistance-dissipated voltage per unit
        # length

        dR_dx = self.rho / self.area
        # Internal resistance per unit length

        return dT_dx, dq_dx, dVs_dx, dV_dx, dR_dx

    def solve_leg(self):

        """Solves leg based on specified convection boundary
        conditions."""
        self.T_h = self.T_h_conv
        self.T_c = self.T_c_conv

        self.fsolve_output = fsolve(self.get_error, x0=self.T_h - 1.)

    def get_error(self, T_h):
```

```
        """Returns error in heat flux and temperature convection
        boundary conditions.

        Methods:
        self.solve_leg_once"""

        self.T_h = T_h[0]

        self.q_h_conv = self.U_hot * (self.T_h_conv - self.T_h)
        self.q_h = self.q_h_conv

        self.solve_leg_once(self.q_h)

        self.q_c_conv = self.U_cold * (self.T_c - self.T_c_conv)

        self.q_c_error = self.q_c - self.q_c_conv

        return self.q_c_error

    def solve_leg_once(self, q_h):

        """Solves leg once based on hot side heat flux.

        Solution procedure comes from Ch. 12 of Thermoelectrics
        Handbook, CRC/Taylor & Francis 2006. x-axis has been reversed
        relative to reference procedure.

        Inputs:
        q_h - hot side heat flux (W / m^2)

        """

        self.q_h = q_h
        self.y0 = np.array([self.T_h, self.q_h, 0, 0, 0])

        self.y = odeint(self.get_dTq_dx, y0=self.y0, t=self.x)

        self.T_x = self.y[:, 0]
        self.q_x = self.y[:, 1]
        self.Vs_x = self.y[:, 2]
        self.V_x = self.y[:, 3]
        self.R_int_x = self.y[:, 4]

        self.T_c = self.T_x[-1]
        self.q_c = self.q_x[-1]

        self.Vs = self.Vs_x[0] - self.Vs_x[-1]
        self.V = self.V_x[0] - self.V_x[-1]
        self.R_internal = self.R_int_x[-1]

        self.P_flux = self.J * self.V
        self.P = self.P_flux * self.area
        # Power for the entire leg (W)

        self.eta = self.P / (self.q_h * self.area)
        # Efficiency of leg
        self.R_load = self.V / self.I

        # Sanity check.  q_h - q_c should be nearly equal but not
        # exactly equal to P.  It is not exact because of spatial
        # asymmetry in electrical resistivity along the leg.  I
        # imported assert_approx_equal in the front matter to make
        # this print an error if there is too much disagreement.
```

```
        self.P_from_heat = (self.q_h - self.q_c) * self.area

        sig_figs = 3
        # tolerance in number of sig figs that agree.  Higher number
        # is stricter aka tighter tolerance.  This may need to be
        # reduced if you know the code is correct and it is still
        # printing the error statement.

        try:
            assert_approx_equal(self.P, self.P_from_heat, sig_figs)
        except AssertionError:
            print "\nPower from q_h - q_c and I  **  2 * R disagree."
            print "Consider reducing sig_figs under solve_leg_once"
            print "in leg.py if you think this is an error."

    def set_q_guess(self):

        """Sets guess for q_c to be used by iterative solutions.

        Methods:

        self.set_TEproperties(T_props)

        """

        self.T_props = 0.5 * (self.T_h + self.T_c)
        self.set_TEproperties(T_props=self.T_props)
        delta_T = self.T_h - self.T_c
        self.q_c = - (
            self.alpha * self.T_c * self.J - delta_T / self.length *
        self.k - self.J ** 2 * self.length * self.rho
            )
        # cold side heat flux (W / (m^2 * K))

        self.q_h = - (
            self.alpha * self.T_h * self.J - delta_T / self.length *
            self.k + self.J ** 2. * self.length * self.rho / 2.
            )

        self.q_c_guess = self.q_c
        # cold side heat flux (W / (m^2 * K))
        self.q_h_guess = self.q_h
        self.q_guess = self.q_h

    def solve_leg_anal(self):

        """Analytically solves the leg based on lumped properties.

        Methods:

        self.set_TEproperties

        No iteration is needed.

        """

        self.T_props = 0.5 * (self.T_h + self.T_c)

        self.set_TEproperties(T_props=self.T_props)

        delta_T = self.T_h - self.T_c
        self.q_h = (
```

```
            self.alpha * self.T_h * self.J - delta_T / self.length *
            self.k + self.J ** 2. * self.length * self.rho / 2.
            )
        self.q_c = (
            self.alpha * self.T_c * self.J - delta_T / self.length *
            self.k - self.J ** 2 * self.length * self.rho
            )

        self.P_flux = (
            (self.alpha * delta_T * self.J + self.rho * self.J ** 2 *
             self.length)
            )
        self.P = self.P_flux * self.area
        self.eta = self.P / (self.q_h * self.area)
        self.eta_check = (
            (self.J * self.alpha * delta_T + self.rho * self.J **
            2. * self.length) / (self.alpha * self.T_h * self.J -
            delta_T / self.length * self.k + self.J ** 2 * self.length
            * self.rho / 2.)
            )
        self.V = -self.P / np.abs(self.I)
        self.R_internal = self.rho * self.length / self.area
        self.R_load = - self.V / self.I

    def set_ZT(self):

        """Sets ZT based on formula.

        Formula:

        self.ZT = self.sigma * self.alpha ** 2. / self.k

        """

        self.ZT = self.alpha ** 2. * self.T_props / (self.k * self.rho)

    def set_power_factor(self):

        """Sets power factor and maximum theoretical power for leg."""

        self.power_factor = self.alpha ** 2 * self.sigma
        self.power_max = (
            self.power_factor * self.T_props ** 2 / self.length *
            self.area
            )

    def solve_leg_transient(self):

        """Solves leg based on array of transient BC's."""

        self.delta_x = self.x[1] - self.x[0]

        self.y0 = self.T_x

        try:
            self.T_xt

        except AttributeError:
            self.odeint_output = odeint(
                self.get_dTx_dt, y0=self.y0, t=self.t_array,
                full_output=1
                )
            self.T_xt = self.odeint_output[0]
```

```
    else:
        self.y0 = self.T_xt[-1,:]
        self.odeint_output = odeint(
            self.get_dTx_dt, y0=self.y0, t=self.t_array,
            full_output=1
            )
        self.T_xt = np.concatenate((self.T_xt, self.odeint_output[0]))

def get_dTx_dt(self, T, t):

    """Returns derivative of array of T wrt time.
    """

    self.dT_dt = np.zeros(T.size)
    self.q0 = np.zeros(T.size)
    self.dq_dx_ss = np.zeros(T.size)
    self.dq_dx = np.zeros(T.size)
    self.dT_dx = np.zeros(T.size)

    self.dT_dx[1:-1] = 0.5 * (T[2:] - T[:-2]) / self.delta_x
    self.dT_dx[0] = (T[1] - T[0]) / self.delta_x
    self.dT_dx[-1] = (T[-1] - T[-2]) / self.delta_x

    for i in range(self.nodes):

        T_props = T[i]  # i for central differencing
        self.set_TEproperties(T_props)
        self.set_ZT()

        self.q0[i] = (
            self.J * T[i] * self.alpha - self.k * self.dT_dx[i]
            )

        self.dq_dx_ss[i] = (
            (self.rho * self.J ** 2. * (1. + self.ZT)) - self.J *
            self.alpha * self.q0[i] / self.k
            )

    # hot side BC, q_h
    self.q0[0] = self.U_hot * (self.T_h_conv - T[0])

    # cold side BC, q_c
    self.q0[-1] = self.U_cold * (T[-1] - self.T_c_conv)

    self.dq_dx[1:-1] = (
        (self.q0[2:] - self.q0[:-2]) / (2. * self.delta_x)
        )
    self.dq_dx[0] = (
        (self.q0[1] - self.q0[0]) / self.delta_x
        )
    self.dq_dx[-1] = (
        (self.q0[-1] - self.q0[-2]) / self.delta_x
        )

    for i in range(self.nodes):

        T_props = T[i]  # i for central differencing
        self.set_TEproperties(T_props)
        self.set_ZT()

        self.dT_dt[i] = (
            1. / self.C * (-self.dq_dx[i] + self.dq_dx_ss[i])
```

```
                )

        return self.dT_dt
```

Listing E.9: mat_prop.py

```
"""Module containing set properties function."""


import numpy as np


def import_raw_property_data(self):

    """Imports and sets values for material properties as a function
    of temperature.  These values come from literature, and they may
    come from experiments or curve fitting in the future.

    """

    print "running import_raw_property_data"

    if self.material == "marlow p-type":
        # added on 10/03/2012
        # we measured Seebeck coefficient for n and p-type
        # sigma and k are from literature right now
        # sigma and k are better performance than actual Marlow
        poly_deg = 3

        self.alpha_raw = np.array([[305.88834, 184.4201],
                                   [315.955, 180.9814],
                                   [325.96334, 183.9347],
                                   [335.91166, 187.8893],
                                   [346.045, 184.0316],
                                   [356.065, 185.297],
                                   [365.95001, 188.8359],
                                   [375.9667, 182.5712],
                                   [385.95, 187.3913],
                                   [396.0383, 185.5793],
                                   [405.9867, 179.1448],
                                   [415.9667, 178.0592],
                                   [426.0117, 180.6971],
                                   [435.995, 170.9204],
                                   [446.0167, 175.7425],
                                   [455.95, 165.7763],
                                   [465.9333, 165.8615],
                                   [476.0367, 158.0667],
                                   [485.955, 145.0191],
                                   [495.885, 144.997],
                                   [505.9884, 133.7025],
                                   [515.995, 125.7248],
                                   [525.9117, 122.7499],
                                   [535.98, 115.2979],
                                   [545.935, 105.3918],
                                   [555.855, 104.7148],
                                   [565.9833, 101.7378],
                                   [575.9683, 95.72422]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[299.5228426396, 1.3873417722],
                               [321.8578680203, 1.3265822785],
```

```
                        [345.5888324873, 1.3164556962],
                        [369.3197969543, 1.3569620253],
                        [391.654822335, 1.3873417722],
                        [416.781725888, 1.4683544304],
                        [440.512690355, 1.6],
                        [462.847715736, 1.7620253165],
                        [474.015228426, 1.8329113924],
                        [497.746192893, 2.035443038],
                        [522.873096447, 2.2075949367]])
        self.k_params = np.polyfit(
            self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
            )

        self.sigma_raw = np.array([[299.4305754926, 9.746835443],
                            [323.4029100874, 8.6835443038],
                            [344.5312214537, 7.5443037975],
                            [369.9479968681, 6.7088607595],
                            [392.5476641, 6.0253164557],
                            [416.58280047, 5.4936708861],
                            [440.626908521, 5.0379746835],
                            [463.271434164, 4.7341772152],
                            [473.158227848, 4.4303797468],
                            [497.229250946, 4.2025316456],
                            [522.744714864, 4.2025316456]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
            )

    if self.material == "marlow n-type":
        # added on 10/03/2012
        # we measured Seebeck coefficient for n and p-type
        # sigma and k are from literature right now
        # sigma and k are better performance than actual Marlow
        poly_deg = 3

        self.alpha_raw = np.array([[305.88834, 184.4201],
                                   [316.02167, -174.2899],
                                   [325.995, -169.4341],
                                   [335.97834, -168.759],
                                   [346.02834, -170.3785],
                                   [355.98333, -172.1414],
                                   [365.98334, -174.0112],
                                   [375.9, -169.3056],
                                   [385.9517, -171.4511],
                                   [395.9383, -165.3796],
                                   [405.905, -163.3501],
                                   [415.95, -159.9725],
                                   [425.9117, -161.7775],
                                   [435.9783, -151.9636],
                                   [445.9483, -156.6737],
                                   [455.9667, -146.8893],
                                   [465.905, -142.9715],
                                   [475.9833, -135.9831],
                                   [485.8583, -133.8981],
                                   [496.005, -130.982],
                                   [505.8667, -127.4579],
                                   [515.9284, -121.2369],
                                   [525.9783, -118.1372],
                                   [535.915, -109.9285],
                                   [545.955, -105.78],
                                   [555.9167, -105.4163],
                                   [565.905, -100.953],
                                   [575.9317, -96.50673],
```

```
                                [575.9384, -97.34524]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[299.5228426396, 1.3873417722],
                               [321.8578680203, 1.3265822785],
                               [345.5888324873, 1.3164556962],
                               [369.3197969543, 1.3569620253],
                               [391.654822335, 1.3873417722],
                               [416.781725888, 1.4683544304],
                               [440.512690355, 1.6],
                               [462.847715736, 1.7620253165],
                               [474.015228426, 1.8329113924],
                               [497.746192893, 2.035443038],
                               [522.873096447, 2.2075949367]])
        self.k_params = np.polyfit(
            self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
            )

        self.sigma_raw = np.array([[299.4305754926, 9.746835443],
                                   [323.4029100874, 8.6835443038],
                                   [344.5312214537, 7.5443037975],
                                   [369.9479968681, 6.7088607595],
                                   [392.5476641, 6.0253164557],
                                   [416.58280047, 5.4936708861],
                                   [440.626908521, 5.0379746835],
                                   [463.271434164, 4.7341772152],
                                   [473.158227848, 4.4303797468],
                                   [497.229250946, 4.2025316456],
                                   [522.744714864, 4.2025316456]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
            )

    if self.material == "BiTe variable n-type":
        poly_deg = 3

        self.alpha_raw = np.array([[112.301006188, -43.0457272427],
                                   [137.129765223, -53.5256743351],
                                   [156.431574472, -62.2549482734],
                                   [179.857338561, -73.5981860267],
                                   [210.184704048, -87.5634182059],
                                   [239.165761255, -98.9227171808],
                                   [272.270773301, -112.895979971],
                                   [308.195946903, -124.275355473],
                                   [339.954650668, -135.642685058],
                                   [373.158864377, -143.544806084]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[100, 29.467689848],
                               [130, 27.8005198844],
                               [160, 26.4696917156],
                               [190, 24.9615826351],
                               [220, 23.1842086837],
                               [250, 21.0620306825],
                               [280, 18.4187680451],
                               [310, 14.8860234335],
                               [340, 9.7526188408],
                               [370, 1.6249327924]])
        self.k_params = np.polyfit(self.k_raw[:, 0], self.k_raw[:, 1], poly_deg)
```

```
        self.sigma_raw = np.array([[109.722222222, 64.802436126],
                                   [125, 59.3315508021],
                                   [138.888888889, 52.8995840761],
                                   [158.333333333, 46.7825311943],
                                   [179.166666667, 41.3057040998],
                                   [201.388888889, 36.1482471777],
                                   [229.166666667, 30.3431372549],
                                   [259.722222222, 25.4976232917],
                                   [284.722222222, 22.2623291741],
                                   [309.722222222, 19.3478906714],
                                   [336.111111111, 16.7528223411],
                                   [359.722222222, 15.1232917409],
                                   [376.388888889, 13.8220439691]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
            )

    if self.material == "BiTe variable p-type":
        # properties trial 2 - need to add a comment
        poly_deg = 3

        self.alpha_raw = np.array([[112.301006188, 43.0457272427],
                                   [137.129765223, 53.5256743351],
                                   [156.431574472, 62.2549482734],
                                   [179.857338561, 73.5981860267],
                                   [210.184704048, 87.5634182059],
                                   [239.165761255, 98.9227171808],
                                   [272.270773301, 112.895979971],
                                   [308.195946903, 124.275355473],
                                   [339.954650668, 135.642685058],
                                   [373.158864377, 143.544806084]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[100, 29.467689848],
                               [130, 27.8005198844],
                               [160, 26.4696917156],
                               [190, 24.9615826351],
                               [220, 23.1842086837],
                               [250, 21.0620306825],
                               [280, 18.4187680451],
                               [310, 14.8860234335],
                               [340, 9.7526188408],
                               [370, 1.6249327924]])
        self.k_params = np.polyfit(self.k_raw[:, 0], self.k_raw[:, 1], poly_deg)

        self.sigma_raw = np.array([[109.722222222, 64.802436126],
                                   [125, 59.3315508021],
                                   [138.888888889, 52.8995840761],
                                   [158.333333333, 46.7825311943],
                                   [179.166666667, 41.3057040998],
                                   [201.388888889, 36.1482471777],
                                   [229.166666667, 30.3431372549],
                                   [259.722222222, 25.4976232917],
                                   [284.722222222, 22.2623291741],
                                   [309.722222222, 19.3478906714],
                                   [336.111111111, 16.7528223411],
                                   [359.722222222, 15.1232917409],
                                   [376.388888889, 13.8220439691]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
```

```
        )

if self.material == "typical BiTe n-type":
    # Extracted from Bed Poudel et al, Science 320, 634 (2008)
    # This was the properties used for first trial of validation
    # process.
    poly_deg = 3

    self.alpha_raw = np.array([[297.3450032873, -213.717948718],
                           [321.1747205786, -223.974358974],
                           [343.6845825115, -227.820512821],
                           [367.6137409599, -231.025641026],
                           [391.61522025, -229.102564103],
                           [414.22452334, -225.897435897],
                           [439.726660092, -217.564102564],
                           [462.462524655, -205.384615385],
                           [472.47896121, -195.128205128],
                           [496.72452334, -175.897435897],
                           [522.425542406, -153.46153846]])
    self.alpha_params = np.polyfit(
        self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
        )

    self.k_raw = np.array([[299.5228426396, 1.3873417722],
                       [321.8578680203, 1.3265822785],
                       [345.5888324873, 1.3164556962],
                       [369.3197969543, 1.3569620253],
                       [391.654822335, 1.3873417722],
                       [416.781725888, 1.4683544304],
                       [440.512690355, 1.6],
                       [462.847715736, 1.7620253165],
                       [474.015228426, 1.8329113924],
                       [497.746192893, 2.035443038],
                       [522.873096447, 2.2075949367]])
    self.k_params = np.polyfit(
        self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
        )

    self.sigma_raw = np.array([[299.4305754926, 9.746835443],
                           [323.4029100874, 8.6835443038],
                           [344.5312214537, 7.5443037975],
                           [369.9479968681, 6.7088607595],
                           [392.5476641, 6.0253164557],
                           [416.58280047, 5.4936708861],
                           [440.626908521, 5.0379746835],
                           [463.271434164, 4.7341772152],
                           [473.158227848, 4.4303797468],
                           [497.229250946, 4.2025316456],
                           [522.744714864, 4.2025316456]])
    self.sigma_params = np.polyfit(
        self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
        )

if self.material == "typical BiTe p-type":

    poly_deg = 3
    # Extracted from Bed Poudel et al, Science 320, 634 (2008)
    # This was the properties used for first trial of validation
    # process.
    self.alpha_raw = np.array([[297.3450032873, 213.717948718],
                           [321.1747205786, 223.974358974],
                           [343.6845825115, 227.820512821],
                           [367.6137409599, 231.025641026],
```

```
                            [391.61522025, 229.102564103],
                            [414.22452334, 225.897435897],
                            [439.726660092, 217.564102564],
                            [462.462524655, 205.384615385],
                            [472.47896121, 195.128205128],
                            [496.72452334, 175.897435897],
                            [522.425542406, 153.46153846]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[299.5228426396, 1.3873417722],
                           [321.8578680203, 1.3265822785],
                           [345.5888324873, 1.3164556962],
                           [369.3197969543, 1.3569620253],
                           [391.654822335, 1.3873417722],
                           [416.781725888, 1.4683544304],
                           [440.512690355, 1.6],
                           [462.847715736, 1.7620253165],
                           [474.015228426, 1.8329113924],
                           [497.746192893, 2.035443038],
                           [522.873096447, 2.2075949367]])
        self.k_params = np.polyfit(
            self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
            )

        self.sigma_raw = np.array([[299.4305754926, 9.746835443],
                            [323.4029100874, 8.6835443038],
                            [344.5312214537, 7.5443037975],
                            [369.9479968681, 6.7088607595],
                            [392.5476641, 6.0253164557],
                            [416.58280047, 5.4936708861],
                            [440.626908521, 5.0379746835],
                            [463.271434164, 4.7341772152],
                            [473.158227848, 4.4303797468],
                            [497.229250946, 4.2025316456],
                            [522.744714864, 4.2025316456]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
            )

    if self.material == "HMS":
        # Raw data taken from Luo et al. HMS is p-type
        poly_deg = 3
        # print "Curve fitting for HMS"

        self.alpha_raw = np.array([[296.89119171, 138.265544041],
                            [380.829015544, 140.620466321],
                            [561.139896373, 176.845854922],
                            [701.03626943, 206.270725389],
                            [806.735751295, 217.652849741],
                            [900., 205.769430052]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[300, 2.40620446533],
                           [485.869565217, 2.20460634548],
                           [593.47826087, 2.1252173913],
                           [707.608695652, 2.07168037603],
                           [815.217391304, 2.09607520564],
                           [900.0, 2.12944770858]])
        self.k_params = np.polyfit(
```

```
            self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
            )

        self.sigma_raw = np.array([[283.888641142, 6.55346563038],
                                   [396.056571319, 6.22507485507],
                                   [573.510861948, 4.86979996178],
                                   [786.035548194, 3.5398961585],
                                   [856.520354208, 3.34810791871],
                                   [901.20405173, 3.34610116583]])
        self.sigma_params = np.polyfit(
            self.sigma_raw[:, 0], self.sigma_raw[:, 1], poly_deg
            )

    if self.material == "MgSi":
        # Raw data comes from Gao et al. MgSi is n-type
        poly_deg = 2
        # print "Curve fitting for MgSi"

        self.alpha_raw = np.array([[311.289993567, -111.872146119],
                                   [464.006967001, -141.552511416],
                                   [644.121200709, -184.931506849],
                                   [777.984904831, -207.762557078]])
        self.alpha_params = np.polyfit(
            self.alpha_raw[:, 0], self.alpha_raw[:, 1], poly_deg
            )

        self.k_raw = np.array([[291.236965464, 2.80871520138],
                               [472.020791479, 2.62097005644],
                               [725.982971396, 2.38897924041],
                               [576.615963519, 2.50282215632]])
        self.k_params = np.polyfit(
            self.k_raw[:, 0], self.k_raw[:, 1], poly_deg
            )

        self.sigma_raw = np.array([[307.385007162, 13.156135604],
                                   [456.638548464, 9.79627566449],
                                   [574.442145472, 8.21502466974],
                                   [722.524271845, 7.17849753303]])
        self.sigma_params = np.polyfit(self.sigma_raw[:, 0], self.sigma_raw[:, 1],
                                   poly_deg)


def set_properties_v_temp(self, T_props):

    """ Sets properties based on polynomial fit values.

    Used by set_TEproperties to set the temperature-dependent
    properties of materials for which polynomial curve fits have been
    done.

    This may need to changed to spline fitting at some point for a
    more accurate fit.

    Inputs:

    T_props : temperature (K) at which properties are to be evaluated

    """
    try:
        self.alpha_params
    except AttributeError:
        self.import_raw_property_data()
    self.alpha = (np.polyval(self.alpha_params, T_props) * 1.e-6)
```

```
        # Seebeck coefficient (V/K)
        self.k = np.polyval(self.k_params, T_props)
        # thermal conductivity (W/m-K)
        self.sigma = np.polyval(self.sigma_params, T_props) * 1.e4
        # electrical conductivity (S/m)
        self.rho = 1. / self.sigma
        # electrical resistivity (Ohm-m)


    def set_TEproperties(self, T_props):

        """Sets TE properties

        Inputs:

        T_props : temperature (K) at which properties are to be evaluated

        This method exists to separater materials with constant properties
        from materials with temperature dependent properties.  It uses
        set_properties_v_temp for the latter type of materials.

        """

        self.T_props = T_props

        # Materials with tabulated properties
        if self.material == 'HMS':
            self.set_properties_v_temp(T_props)

        elif self.material == 'MgSi':
            self.set_properties_v_temp(T_props)

        elif self.material == 'BiTe variable n-type':
            self.set_properties_v_temp(T_props)

        elif self.material == 'BiTe variable p-type':
            self.set_properties_v_temp(T_props)

        elif self.material == 'typical BiTe n-type':
            self.set_properties_v_temp(T_props)

        elif self.material == 'typical BiTe p-type':
            self.set_properties_v_temp(T_props)

        elif self.material == 'marlow p-type':
            self.set_properties_v_temp(T_props)

        elif self.material == 'marlow n-type':
            self.set_properties_v_temp(T_props)

        # Material properties for validation trial. These properties are
        # for typical BiTe materials at 423 K. The properties for a range
        # of temperature are given above as a poly curve.
        elif self.material == 'BiTe variable n-type':
            self.k = 1.54  # Thermal conductivity (W/m-K)
            self.alpha = -150.e-6  # Seebeck coefficient (V/K)
            # I made this negative even though it's for a p-type
            # material.  This is just for a hypothetical model.
            self.rho = 9.0909 * 1.e-6
            # electrical resistivity (Ohm-m)

        elif self.material == 'BiTe variable p-type':
            self.k = 1.54  # Thermal conductivity (W/m-K)
```

```
        self.alpha = 150.e-6  # Seebeck coefficient (V/K)
        self.rho = 9.0909 * 1.e-6
        # electrical resistivity (Ohm-m)

    # Materials with properties that are either constant or dependent
    # on temperature by some convenient function.
    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex1 n-type':
        self.k = 54. / T_props * 100.
        # thermal conductivity (W/m-K)
        self.alpha = (0.268 * T_props - 329.) * 1.e-6
        # Seebeck coefficient (V/K)
        self.sigma = (T_props - 310.) / 0.1746
        # electrical conductivity (S/cm)
        self.rho = 1. / self.sigma / 100.
        # electrical resistivity (Ohm-m)

    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex1 p-type':
        self.k = 3.194 / T_props * 100.
        # thermal conductivity (W/m-K)
        self.alpha = (0.150 * T_props + 211.) * 1.e-6
        # Seebeck coefficient (V/K)
        self.sigma = 25.
        # electrical conductivity (S/cm)
        self.rho = 1. / self.sigma / 100.
        # electrical resistivity (Ohm-m)

    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex2 n-type':
        self.k = 3. / T_props * 100.
        # thermal conductivity (W/m-K)
        self.alpha = (0.20 * T_props - 400.) * 1.e-6
        # Seebeck coefficient (V/K)
        self.sigma = 1.e5 / T_props
        # electrical conductivity (S/cm)
        self.rho = 1. / self.sigma / 100.
        # electrical resistivity (Ohm-m)

    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex2 p-type':
        self.k = 10. / T_props * 100.
        # thermal conductivity (W/m-K)
        self.alpha = (200.) * 1.e-6
        # Seebeck coefficient (V/K)
        self.sigma = T_props
        # electrical conductivity (S/cm)
        self.rho = 1. / self.sigma / 100.
        # electrical resistivity (Ohm-m)

    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex3 n-type':
        self.k = 3. / T_props * 100.
        # thermal conductivity (W/m-K)
        self.alpha = 0.20 * T_props * 1.e-6
        # Seebeck coefficient (V/K)
        self.sigma = 1000.
        # electrical conductivity (S/cm)
        self.rho = 1. / self.sigma / 100.
        # electrical resistivity (Ohm-m)

    # from CRC TE Handbook Table 12.1
    elif self.material == 'ex3 p-type':
```

```
            self.k = 10. / T_props * 100.
            # thermal conductivity (W/m-K)
            self.alpha = 200. * 1.e-6
            # Seebeck coefficient (V/K)
            self.sigma = T_props
            # electrical conductivity (S/cm)
            self.rho = 1. / self.sigma / 100.
            # electrical resistivity (Ohm-m)

        # From CRC TE Handbook Table 27.7
        elif self.material == 'constant BiTe n-type':
            self.k = 1.5  # Thermal conductivity (W/m-K)
            self.alpha = -206.e-6  # Seebeck coefficient (V/K)
            # I made this negative even though it's for a p-type
            # material.  This is just for a hypothetical model.
            self.rho = 8.89 * 1.e-6
            # electrical resistivity (Ohm-m)

        # From CRC TE Handbook Table 27.7
        elif self.material == 'constant BiTe p-type':
            self.k = 1.5  # Thermal conductivity (W/m-K)
            self.alpha = 206.e-6  # Seebeck coefficient (V/K)
            self.rho = 8.89 * 1.e-6
            # electrical resistivity (Ohm-m)

        # Alumina with pure conduction
        elif self.material == 'alumina':
            self.k = 1.5  # Thermal conductivity (W/m-K) this needs to be
                          # updated to what it actually is
            self.alpha = 1.e-9  # Seebeck coefficient (V/K)
            # I made this negative even though it's for a p-type
            # material.  This is just for a hypothetical model.
            self.rho = 1.
            # dummy value since I don't know what it
            # actually is and it doesn't matter.
            # electrical resistivity (Ohm-m)

        # Direct contact between hot and cold side
        elif self.material == 'none':
            self.k = 1.e9  # really high thermal condutivity (W/m-K) so
                           # that resistance is zero
            self.alpha = 1.e-9  # dummy value
            self.rho = 1.
            # dummy value
```

Listing E.10: exhaust.py

```
"""Contains class for exhaust side of heat exchanger"""

# In python directory
import properties as prop
reload(prop)

# In this directory
import functions
reload(functions)
import enhancement
reload(enhancement)


class Exhaust(prop.ideal_gas):

    """Class for engine exhaust in heat exchanger.
```

```
    Methods:

    __init__
    set_fluid_props

    """

    def __init__(self):

        """
        Sets a bunch of constants, binds methods, inits parent class

        self.enh_lib = enhancement - Used in hx.py

        Also initializes super class, which is ideal_gas from the
        properties script.  I keep this script in ~/Documents/Python,
        which is part of my python path.

        """

        super(Exhaust, self).__init__()

        self.enh_lib = enhancement
        self.enh = None
        self.T_ref = 300.
        self.P = 101.
        self.height = 1.5e-2
        self.ducts = 1
        self.sides = 2
        self.mdot_omega = 0.2 / 60.

        self.Nu_coeff = 0.023

        functions.bind_functions(self)

    def set_fluid_props(self):

        """
        Sets properties needed for set_flow.

        Methods:

        self.set_thermal_props

        """

        self.set_thermal_props()
        self.c_p = self.c_p_air
        self.k = self.k_air
```

Listing E.11: engine.py

```
"""Contains Engine class used to determine attributes of engine.  This will be
fleshed out with experimental data later."""

# User defined modules
import properties as prop

class Engine(object):

    """Class definition for engine object.
```

```
    Methods:

    self.set_mdot_charge

    """

    def __init__(self,**kwargs):

        """Sets constants

        Methods:

        self.air.set_TempPress_dependents()

        Instantiated in hx.py in HX class

        """

        if 'RPM' in kwargs:
            self.RPM = kwargs['kwargs']
        else:
            self.RPM = 2000. # engine speed (RPM)
        if 'torque' in kwargs:
            self.torque = kwargs['torque']
        else:
            self.torque = 300. # brake torque (lb-ft)
        self.displacement = 6.7e-3 # engine swept displacement (m**3)
        self.cylinders = 6. # number of cylinders
        self.eta_V = 1. # volumetric efficiency of engine. Can exceed unity for
        # turbo-charged unthrottled engine. Accounts for error
        # in intake manifold pressure estimate.
        self.T_intake = 300. # engine intake temperature (K)
        self.P_intake = 101.325 # pressure (kPa) at intake manifold
        self.air = prop.ideal_gas() # engine working fluid is ideal gas with
        # the properties of air
        self.air.T = self.T_intake
        self.air.P = self.P_intake
        self.air.set_TempPres_dependents()

    def set_mdot_charge(self):

        """Sets charge mass flow rate."""

        self.mdot_charge =(
            (self.RPM / 2. * self.displacement * self.eta_V *
            self.air.rho) / 60.
            )
        # charge flow (kg/s) in engine
```

Listing E.12: coolant.py

```
"""Contains class for coolant side of heat exchanger."""

# In local directory
import functions
reload(functions)
import enhancement
reload(enhancement)


class Coolant(object):
    """
    class for coolant flow
```

```
    Methods:

    __init__
    set_fluid_props

    """

    def __init__(self):
        """
        Sets contants and instantiates classes.

        self.enh_lib = enhancement - Used in hx.py

        """

        self.enh_lib = enhancement
        self.enh = None

        self.height = 1.e-2
        # height (m) of coolant duct
        self.mdot = 1.0
        # mass flow rate (kg/s) of coolant
        self.ducts = 2 # number of coolant ducts per hot duct
        self.geometry = 'parallel plates'
        self.c_p = 4.179
        # Specific heat (kJ/kg*K) of water at 325K
        self.mu = 5.3e-4
        # viscosity of water at 325K (Pa*s), WolframAlpha
        self.k = 0.646e-3
        # thermal conductivity of water at 325K (kW/m*K) through
        # cooling duct
        self.Pr = (7.01 + 5.43)/2 # Prandtl # of water from Engineering
        # Toolbox
        self.rho = 1000.
        # density (kg/m**3) of water
        self.Nu_coeff = 0.023
        self.enthalpy0 = 113.25
        # enthalpy (kJ/kg) of coolant at restricted dead state
        self.entropy0 = 0.437
        # entropy (kJ/kg*K) of coolant at restricted dead state
        self.sides = 1

        functions.bind_functions(self)

    def set_fluid_props(self):

        """Sets fluid properties needed for set_flow."""

        self.nu = self.mu / self.rho
```

Listing E.13: enhancement.py

```
# coding=utf-8
"""Contains classes for modeling convection heat transfer
enhancement."""

# Distribution libraries
import numpy as np


class BejanPorous(object):
```

```
    """Class for porous media according to the book of Bejan.

    Bejan, A.  Designed Porous Media: Maximal Heat Transfer Density at
    Decreasing Length Scales.  International Journal of Heat and Mass
    Transfer 47, no. 14 (2004): 3073  3083.

    Methods:

    __init__
    solve_enh

    """

    def __init__(self):

        """Initializes a bunch of constants."""

        self.porosity = 0.92
        self.k_matrix = 5.8e-3
        self.PPI = 10.
        self.K = 2.e-7
        self.Nu_D
        # Nu for porous media parallel plates with const heat flux.
        # Bejan Eq. 12.77

    def solve_enh(self):

        """Solves for convection parameters with enhancement."""

        self.Re_K = self.velocity * self.K ** 0.5 / self.nu
        # Re based on permeability from Bejan Eq. 12.11
        self.f = 1. / self.Re_K + 0.55
        # Darcy Law, Bejan Eq. 12.14.  It turns out that f is pretty
        # close to 0.55
        self.k = self.k_matrix
        self.deltaP = (self.f * self.perimeter * self.node_length /
        self.flow_area * (0.5 * self.rho * self.velocity ** 2) *
        0.001)
        # pressure drop (kPa)
        self.h_conv = self.Nu_D * self.k / self.D
        # coefficient of convection (kW/m^2-K)


class MancinPorous(object):

    """Class for modeling porous media according to Mancin.

    Mancin, S., C. Zilio, A. Cavallini, and L. Rossetto.  Pressure
    Drop During Air Flow in Aluminum Foams.  International Journal of
    Heat and Mass Transfer 53, no. 15  16 (2010): 3121  3130.

    Methods:

    __init__
    solve_enh

    """

    def __init__(self, flow):

        """Sets constants."""

        self.flow = flow
```

```
        self.porosity = 0.92
        self.k_matrix = 5.8e-3
        self.PPI = 10.
        self.k = self.k_matrix
        self.Nu_D = 4.93
        # Nu for porous media parallel plates with T_w = const.  Bejan
        # Eq. 12.77

    def solve_enh(self):

        """Solves for convection parameters with enhancement."""

        self.G = self.flow.rho * self.flow.velocity
        # Mass velocity from Mancin et al.
        self.D_pore = 0.0122 * self.PPI ** (-0.849)
        # hydraulic diameter (m) of porous media based on Mancin et
        # al.
        self.Re_D = (
            self.D_pore * self.G / (self.flow.mu * self.porosity)
            )
        # Re of porous media from Mancin et al.
        self.F = (
            (1.765 * self.Re_D ** (-0.1014) * self.porosity ** 2. /
        self.PPI ** (0.6))
            )
        # friction factor from Mancin et al.
        self.flow.f = self.F
        # possibly wrong assignment but gets code to shut up and run
        self.flow.deltaP = (
            self.flow.length * 2. * self.F * self.G ** 2. / (self.D_pore *
        self.flow.rho) * 0.001
            )
        # pressure drop from Mancin et al.
        self.flow.h_conv = (
            self.flow.Nu_D * self.k_matrix / self.flow.D
            )
        # coefficient of convection (kW/m^2-K)


class IdealFin(object):

    """Class for modeling straight fin.

    Mills, A. F. Heat Transfer. 2nd ed. Prentice Hall, 1998.

    Methods:

    __init__
    set_eta
    set_enh_geometry
    set_h_and_P
    solve_enh

    """

    def __init__(self, flow):

        """Sets constants and things are needed at runtime.  Runs
        set_fin_height and set_area_convection."""

        self.type = 'IdealFin'
        self.thickness = 1.e-3
        # fin thickness (m)
```

```
        self.k = 0.2
        # thermal conductivity (kW / (m * K)) of fin material
        self.spacing = 0.003
        # distance (m) between adjacent fin edges

        self.flow = flow
        self.set_fin_height()

    def set_fin_height(self):

        """Sets fin height based on half of duct height."""

        if self.flow.sides == 2:
            self.height = self.flow.height / 2
            # height of fin pair such that their tips meet in the
            # middle and are adiabatic.
        else:
            self.height = self.flow.height
            # height of fin that crosses the channel

    def set_area_convection(self):

        """Sets finned and unfinned area for convection."""

        if self.flow.sides == 2:
            self.flow.area_unfinned = (
                2. * (self.flow.width - self.N * self.thickness)
                )
            # unfinned base area on both sides of duct
            self.flow.area_finned = 2. * self.N * self.thickness
            # finned base area on both sides of duct
        else:
            self.flow.area_unfinned = (
                self.flow.width - self.N * self.thickness
                )
            # unfinned base area on both sides of duct
            self.flow.area_finned = self.N * self.thickness
            # finned base area on both sides of duct

    def set_enh_geometry(self):

        """Fixes appropriate geometrical parameters.

        """

        self.set_fin_height()

        self.N = (
            (self.flow.width / self.spacing - 1.) / (1. +
            self.thickness / self.spacing)
            )

        self.flow.perimeter = (
            2. * (self.spacing + self.flow.height) * (self.N + 1.)
            )
        # perimeter of new duct formed by fins with constant overall
        # duct width
        self.flow.flow_area = (
            self.spacing * self.flow.height * (self.N + 1.)
            )
        # flow area (m^2) of new duct formed by fin

        self.flow.flow_area_nominal = (
```

```
            self.flow.height * self.flow.width
            )

        self.flow.sigma = (
            self.flow.flow_area / self.flow.flow_area_nominal
            )

        self.flow.D = 4. * self.flow.flow_area / self.flow.perimeter
        # hydraulic diameter (m)

        self.set_area_convection()

    def set_eta(self):

        """Sets fin efficiency and related parameters.

        """

        self.beta = np.sqrt(
            2. * self.flow.h_conv / (self.k * self.thickness)
            )
        # dimensionless fin parameter
        self.xi = self.beta * self.height
        # beta times fin length (self.height)
        self.eta = np.tanh(self.xi) / self.xi
        # fin efficiency

    def set_h_and_P(self):

        """Sets effective heat transfer coefficient and deltaP.

        """

        self.flow.h_unfinned = self.flow.h_conv

        self.effectiveness = (
            self.eta * 2. * self.height / self.thickness
            )
        self.h_base = self.effectiveness * self.flow.h_conv

        self.flow.h_conv = (
            (self.flow.h_unfinned * self.flow.area_unfinned +
            self.h_base * self.flow.area_finned) / self.flow.width
            )

        self.flow.deltaP = (
            self.flow.f * self.flow.perimeter * self.flow.node_length
            / self.flow.flow_area * (0.5 * self.flow.rho *
            self.flow.velocity ** 2.) * 0.001
            )
        # pressure drop (kPa)

    def solve_enh(self):

        """Runs all the other methods that need to run.

        Methods:

        self.set_enh_geometry
        self.set_eta
        self.set_h_and_P

        """
```

```
        self.flow.set_Re_dependents()
        self.flow.h_conv = self.flow.Nu_D * self.flow.k / self.flow.D
        # coefficient of convection (kW/m^2-K)

        self.set_eta()
        self.set_h_and_P()


class OffsetStripFin(object):

    """Class for modeling offset strip fins.

    Uses correlations from:

    Manglik, Raj M., and Arthur E. Bergles, 'Heat Transfer and
    Pressure Drop Correlations for the Rectangular Offset Strip Fin
    Compact Heat Exchanger', Experimental Thermal and Fluid Science,
    10 (1995), 171-180 <doi:10.1016/0894-1777(94)00096-Q>.

    Methods:

    __init__
    set_params
    set_f
    set_j
    solve_enh

    """

    def __init__(self, flow):

        """Sets constants and things that need to be guessed to
        execute as a standalone model.

        Sets
        ----------
        self.thickness : thickness (m) of fin strip
        self.l : length (m) of fin
        self.spacing : pitch (m) of fin
        self.k : thermal conductivity (W/m/K) of osf material"""

        self.thickness = 0.001
        self.l = 0.01
        self.spacing = 0.001
        self.k = 0.2
        self.flow = flow

    def set_params(self):

        """Sets flow parameters.

        See Manglik and Bergles Fig. 1.

        """

        self.height = self.flow.height - self.thickness
        # vertical gap (m) between fins and hx walls

        # self.spacing : horizontal gap (m) between fins

        self.alpha = self.spacing / self.height
        self.delta = self.thickness / self.l
```

```
        self.gamma = self.thickness / self.spacing

        self.rows = self.flow.length / self.l
        # number of rows of offset strip fins in streamwise direction

        self.area_frac = ((self.spacing * self.height) /
        ((self.height + self.thickness) * (self.spacing +
        self.thickness)))
        # fraction of original area still available for flow

        self.area_enh = ((self.height * self.thickness + self.height
                          * self.l + self.spacing * self.l) /
                         ((self.thickness + self.spacing) * self.l))
        # heat transfer area enhancement factor

        self.D = (4. * self.spacing * self.height * self.l / (2. *
        (self.spacing * self.l + self.height * self.l + self.thickness
        * self.height) + self.thickness * self.spacing))

        self.flow_area = self.flow.flow_area * self.area_frac
        # actual flow area (m^2)
        self.perimeter = 4. * self.flow_area / self.D
        self.velocity = self.flow.velocity / self.area_frac
        # actual velocity (m/s) based on flow area
        self.Re_D = self.velocity * self.D / self.flow.nu

    def set_f(self):

        """Sets friction factor, f."""

        self.f = (
            9.6243 * self.Re_D ** -0.7422 * self.alpha ** -0.1856 *
            self.delta ** 0.3053 * self.gamma ** -0.2659
            )

    def set_j(self):

        """Sets Colburn factor, j."""

        self.j = (
            0.6522 * self.Re_D ** -0.5403 * self.alpha ** -0.1541 *
            self.delta ** 0.1499 * self.gamma ** -0.0678 * (1. +
            5.269e-5 * self.Re_D ** 1.340 * self.alpha ** 0.504 *
            self.delta ** 0.456 * self.gamma ** -1.055) ** 0.1
            )

    def solve_enh(self):
        """Runs all the methods for this class.

        self.h_convcomes from Thermal Design by HoSung Lee, eq. 5.230
        self.eta_fin : fin efficiency


        Methods:

        self.set_params
        self.set_f
        self.set_j

        """

        self.set_params()
        self.set_f()
```

```
        self.flow.f = self.f
        self.flow.deltaP = (self.f * self.perimeter * self.flow.node_length /
                    self.flow.flow_area * (0.5 * self.flow.rho * self.velocity
        ** 2) * 0.001)
        # pressure drop (kPa)
        self.set_j()
        self.h_conv = (
            self.j * self.flow.mdot / self.flow_area * self.flow.c_p /
        self.flow.Pr ** 0.667
            )
        self.beta = np.sqrt(2. * self.h_conv / (self.k * self.thickness))
        self.xi = self.beta * self.height / 2.
        self.eta_fin = np.tanh(self.xi) / self.xi
        self.effectiveness = self.eta_fin * self.height / self.thickness
        self.h_base = self.h_conv * self.effectiveness
        self.flow.h_conv = (
            (self.h_base * self.thickness + self.h_conv *
             self.spacing) / (self.spacing + self.thickness)
            )

        self.flow.Nu_D = self.flow.h_conv * self.flow.D / self.flow.k


class JetArray(object):
    """Class for impinging jet array.

    Huber, Aaron M., and Raymond Viskanta.  Effect of Jet-jet Spacing
    on Convective Heat Transfer to Confined, Impinging Arrays of
    Axisymmetric Air Jets.  International Journal of Heat and Mass
    Transfer 37, no. 18 (December 1994): 2859  2869.
    """

    def __init__(self, flow):
        """Initializes variables that do not require calculation.

        Variables that are set
        ----------------------
        self.D : jet diameter (m)
        self.H : distance (m) from jet exit to impingement surface
        self.K : minor loss coefficient. Fox, McDonald, and Pritchard
        Table 8.2.
        self.spacing : distance (m) between adjacent jets"""

        self.D = 3.0e-3
        self.H = 1.0e-2
        self.K = 0.5
        self.spacing = 0.5e-2
        self.flow = flow

    def set_number(self, flow):
        """Sets number of jets based on jet spacing and overall size
        of jet array.

        Set variables
        -------------
        self.N_streamwise : number of jets in streamwise direction
        self.N_transverse : number of jets in transverse direction
        self.N : total number of jets in array
        self.area : area (m^2) of a jet unit cell

        Variables that must be set to run this method
        ---------
        self.width : width (m) of jet array in transverse direction
```

```
        self.length : length (m) of jet array in streamwise direction"""

        self.N_streamwise = flow.length / self.spacing
        self.N_transverse = flow.width / self.spacing
        self.N = self.N_streamwise * self.N_transverse
        self.area = self.spacing ** 2

    def set_annulus(self, flow):
        """Sets variables related to annulus geometry.

        Requires
        ---------------
        self.width
        self.Vdot

        Sets
        ---------------
        self.ann_area
        self.ann_perimeter
        self.ann_velocity"""

        self.ann_area = flow.width * self.H
        self.ann_perimeter = 2. * (flow.width + self.H)
        self.ann_velocity = flow.Vdot / self.ann_area / 2.

    def set_enh_flow(self, flow):
        """Determines pressure drop through jet array.

        Sets the following variables
        ------------------------------
        self.flow_area : flow area (m^2) for single jet
        self.V : velocity through jet orifice (m/s)
        self.h_loss : head loss (m^2/s^2) through jet orifice
        self.deltaP : pressure drop (kPa) thorugh jet orifice

        Variables that must be set to run this method
        ----------
        self.rho : density (kg/m^3) of fluid passing through jet
        self.Vdot : volume flow rate (m^3/s) of fluid passing through jet
        array"""

        self.flow_area = np.pi * self.D ** 2 / 4.
        self.V = flow.Vdot / (self.flow_area * self.N)
        self.h_loss = self.K * self.V ** 2 / 2.
        flow.deltaP = self.h_loss * flow.rho * 0.001

    def set_Nu_D(self, flow):
        """Sets Nusselt number and some other variables

        Sets the following variables
        ----------------------------
        self.Nu_D : average Nusselt number based on jet diameter
        self.Re_D : Re based on jet diameter

        Variables that must be set to run this method
        ----------
        self.nu : viscosity (m^2 / s) of fluid
        self.Pr : Prandtl number of fluid"""

        self.Re_D = self.V * self.D / flow.nu
        flow.Nu_D = (
            0.285 * self.Re_D ** 0.710 * flow.Pr ** 0.33 * (self.H /
        self.D) ** -0.123 * (self.spacing / self.D) ** -0.725
```

```
            )

    def solve_enh(self):
        """This method is probably not useful so this doc string is
        not good."""

        flow = self.flow
        self.set_number(flow)
        self.set_annulus(flow)
        self.set_enh_flow(flow)
        self.set_Nu_D(flow)
        flow.h_conv = flow.Nu_D * flow.k / self.D
        flow.f = 37.
        # this might need to be changed, or it might be a dummy
        # variable just to keep the code from complaining.
```

Listing E.14: functions.py

```
"""Contains functions to be used in both exhaust and coolant
modules."""

import numpy as np
import types

def set_flow_geometry(self, width):

    """Sets perimeter, flow area, and hydraulic diameter.

    Inputs:

    width (m)

    """

    self.perimeter = 2.*(self.height + width)
    # wetted perimeter (m) of flow
    self.flow_area = self.height * width
    # cross-section area (m^2) of exhaust flow
    self.D = 4. * self.flow_area / self.perimeter
    # coolant hydraulic diameter (m)

    try:
        self.enh
    except AttributeError:
        self.enh = None

    if self.enh != None:
        try:
            self.enh.set_enh_geometry()
        except AttributeError:
            pass
        else:
            self.enh.set_enh_geometry()

def set_Re_dependents(self):

    """Sets Nu and f based on Re.

    Methods:

    self.set_Re

    """
```

```
    self.set_Re()
    # if np.size(self.Re_D) > 1:
    #     if (self.Re_D > 2300.).any():
    #         self.f = 0.078 * self.Re_D**(-1. / 4.)
    #         # friction factor for turbulent flow from Bejan Convection
    #         # Heat Transfer
    #         self.Nu_D = ( self.Nu_coeff * self.Re_D**(4. / 5.) *
    #                       self.Pr**(1. / 3.) )
    #         # Adrian Bejan, Convection Heat Transfer, 3rd ed.,
    #         # Equation 8.30
    #         self.flow = 'turbulent'
    #     else:
    #         self.Nu_D = 7.54 # Bejan, Convection Heat Transfer, Table 3.2
    #             # parallel plates with constant T
    #         self.f = 24. / self.Re_D
    #         self.flow = 'laminar'
    # else:
    if (self.Re_D > 2300.):
        self.f = 0.078 * self.Re_D ** (-1. / 4.) # friction factor for turbulent
        # flow from Bejan
        self.f = self.f * 1.5  # scaled for parallel plates according
        # to Bejan Convection Heat Transfer, 3rd ed. Table 3.2
        if self.sides == 2:
            self.Nu_D = (
                8.235 / 4.364 * self.Nu_coeff * self.Re_D ** (4. / 5.) *
                self.Pr ** (1. / 3.)
                )
            # Adrian Bejan, Convection Heat Transfer, 3rd ed., Equation
            # 8.30, scaled for parallel plates based on Table 3.2
        else:
            self.Nu_D = (
                5.385 / 4.364 * self.Nu_coeff * self.Re_D ** (4. / 5.) *
                self.Pr ** (1. / 3.)
                )
            # Adrian Bejan, Convection Heat Transfer, 3rd ed.,
            # Equation 8.30, scaled for parallel plates with one side
            # adiabatic based on Table 3.2
        self.flow = 'turbulent'
    else:
        if self.sides == 2:
            self.Nu_D = 7.54
            # Bejan, Convection Heat Transfer, Table 3.2, parallel
            # plates with constant T
        else:
            self.Nu_D = 5.385
            # Bejan, Convection Heat Transfer, Table 3.2, parallel
            # plates with constant T, one side adiabatic

        self.f = 24. / self.Re_D
        self.flow = 'laminar'

def set_Re(self):

    """Sets Reynolds number based on hydraulic diameter.

    Requiures:

    self.velocity
    self.D
    self.nu
    self.Re_D = self.velocity * self.D / self.nu
```

```
    """

    self.Re_D = self.velocity * self.D / self.nu
    # Reynolds number

def set_flow(self):

    """
    Sets flow parameters for exhaust or coolant instance.

    See exhaust.py and coolant.py

    Methods
    -------
    self.set_fluid_props
    self.set_Re_dependents
    self.enh.solve_enh
    self.set_thermal_props()

    Used in hx.py by hx.HX.set_convection and possibly
    elsewhere."""

    self.set_fluid_props()

    self.C = self.mdot * self.c_p
    # heat capacity of flow (kW/K)
    self.Vdot = self.mdot / self.rho
    # volume flow rate (m^3/s) of exhaust
    self.velocity = self.Vdot / self.flow_area
    # velocity (m/s) of exhaust

    self.set_Re_dependents()
    self.h_conv= self.Nu_D * self.k / self.D
    # coefficient of convection (kW/m^2-K)

    if self.enh == None:
        self.deltaP = (
            self.f * self.perimeter * self.node_length /
            self.flow_area * (0.5 * self.rho * self.velocity ** 2.) * 0.001
            )
        # pressure drop (kPa)
        print """Something is wrong in set_flow in
        Modules/functions.py if you thought you were using
        enhancement."""
    else:
        self.enh.solve_enh()

    self.Wdot_pumping = self.Vdot * self.deltaP
    # pumping power (kW)

    self.R_thermal = 1. / self.h_conv
    # thermal resistance (m^2-K/kW) of exhaust

def set_enhancement(self, enh_type):

    """For some enhancement strategies, this is useful for
    instantiating the instance because it can pass the exhaust or flow
    instance to the enhancment instance.

    Inputs:

    enh_type - a string describing the enhancemnt type
```

```
    Possible options for enh_type are:
    "IdealFin"
    "OffsetStripFin"
    "MancinPorous"
    "JetArray"
    """

    if enh_type == 'IdealFin':
        self.enh = self.enh_lib.IdealFin(self)

    elif enh_type == 'OffsetStripFin':
        self.enh = self.enh_lib.OffsetStripFin(self)

    elif enh_type == 'MancinPorous':
        self.enh = self.enh_lib.MancinPorous(self)

    elif enh_type == 'JetArray':
        self.enh = self.enh_lib.JetArray(self)

    else:
        print "Error in enh_type specification."
        print "Possible options are:"
        print "IdealFin"
        print "OffsetStripFin"
        print "MancinPorous"
        print "JetArray"

    return self.enh

def bind_functions(self):
    """Binds functions used by both coolant and exhaust."""
    self.set_flow_geometry = (
        types.MethodType(set_flow_geometry, self)
        )
    self.set_Re = (
        types.MethodType(set_Re, self)
        )
    self.set_Re_dependents = (
        types.MethodType(set_Re_dependents, self)
        )
    self.set_flow = (
        types.MethodType(set_flow, self)
        )
    self.set_enhancement = (
        types.MethodType(set_enhancement, self)
        )
```

Listing E.15: platewall.py

```
"""Contains PlateWall class."""

# Created on 7 Nov 2011 by Chad Baker

class PlateWall(object):

    """Class for metal walls of heat exchanger.

    Methods:

    __init__
    set_h

    """
```

```
    def __init__(self):

        """Initializes material properties and geometry defaults."""

        self.k = 200.e-3
        # thermal conductivity (kW/m-K) of Aluminum HX plate
        # (Incropera and DeWitt)
        self.alpha = 73.0e-6
        # thermal diffusivity (m^2/s) of Al HX plate
        self.thickness = 0.00635
        # thickness (m) of HX plate
        self.set_h()

    def set_h(self):

        """Sets the effective convection coefficient."""

        self.h_conv= self.k / self.thickness
        self.R_thermal = 1. / self.h_conv
```

Listing E.16: exp_data.py

```
"""This script imports experimental data and does other stuff with
it."""

import numpy as np
from scipy.optimize import leastsq

import properties as prop


class DataPoint(object):
    pass


class ExpData(object):
    """Class for containing the experimental results."""

    def __init__(self):
        """nothing to do here yet"""

        self.folder = '../../../Heat Exchanger Experiments/gen2/'
        self.file = '2012-09-04'
        self.exh = prop.ideal_gas()
        self.exh.P = 101.325
        self.cool = DataPoint()
        self.set_fit_params()

    def import_data(self):
        """Imports data from csv file as a numpy record array (aka
        structured array)"""

        self.data = np.recfromcsv(self.folder + self.file + '.csv')

        self.exh.T_in = self.data['hx_exh_in_t'] + 273.15  # K
        self.exh.T_out = self.data['hx_exh_out_t'] + 273.15  # K
        self.exh.mdot = self.data['exh_mdot_kgmin'] / 60.  # kg/s
        self.exh.deltaP = (
            self.data['hx_exh_delta_p_2_in_wc'] * 0.249 * 2.  # kPa
            )

        self.cool.T_in = (
```

```
            0.5 * (self.data['hx_cool_1_in_t'] +
            self.data['hx_cool_2_in_t']) + 273.15  # K
            )
        self.cool.T_out = (
            0.5 * (self.data['hx_cool_1_out_t'] +
            self.data['hx_cool_2_out_t']) + 273.15  # K
            )
        self.cool.Vdot = self.data['cool_vdot_gpm']

        self.exh.T_mean = 0.5 * (self.exh.T_in + self.exh.T_out)
        self.exh.deltaT = self.exh.T_in - self.exh.T_out
        self.exh.eta = self.exh.deltaT / (self.exh.T_in - self.cool.T_in)

        self.exh.c_p = np.zeros(self.exh.T_in.size)

        for i in range(self.exh.T_in.size):
            self.exh.T = self.exh.T_mean[i]
            self.exh.set_TempPres_dependents()
            self.exh.c_p[i] = self.exh.c_p_air

        self.exh.Qdot = self.exh.mdot * self.exh.c_p * self.exh.deltaT

    def get_Qdot_fit(self):
        """Uses scipy.optimize leastsq to minimize the error of a
        polynomial in fitting the experimental Qdot data.
        """
        self.leastsq_out = leastsq(
            self.get_Qdot_fit_error, x0=self.fit_params
            )
        self.fit_params = self.leastsq_out[0]

    def set_fit_params(self):
        """Initializes fit parameters."""
        self.fit_params = np.ones(3)

    def eval_Qdot_fit(self, fit_params, mdot, T_in):
        """Evaluates qdot at specific mdot and T_in."""

        self.exh.Qdot_fit = (
            fit_params[0] +
            fit_params[1] * mdot * T_in + fit_params[2] * (mdot *
            T_in) ** 2.
            )

        return self.exh.Qdot_fit

    def rep_Qdot_surf(self, mdot, T_in):
        """Creates 2d surface of Qdot as function of mdot and T_in."""

        self.exh.Qdot_surf = np.zeros([mdot.size, T_in.size])

        for index in np.ndindex(mdot.size, T_in.size):
            i = index[0]
            j = index[1]
            self.exh.Qdot_surf[i, j] = (
                self.eval_Qdot_fit(self.fit_params, mdot[i], T_in[j])
                )

    def get_Qdot_fit_error(self, fit_params):
        """Returns error between statistical fit and experimental Qdot
        data."""

        self.eval_Qdot_fit(fit_params, self.exh.mdot, self.exh.T_in)
```

```
        self.exh.Qdot_fit_err = self.exh.Qdot - self.exh.Qdot_fit

        return self.exh.Qdot_fit_err
```

Listing E.17: real_hx.py

```
"""This script sets up and solves an hx instance for comparison to
experimental data."""

# Distribution Modules
import os
import sys
import numpy as np

# User Defined Modules
cmd_folder = os.path.dirname(os.path.abspath('../Modules/hx.py'))
if cmd_folder not in sys.path:
    sys.path.insert(0, cmd_folder)
import hx
reload(hx)

def get_hx():
    """Sets up and returns an hx instance that has the geometry of the
    experimental hx downstairs."""

    hx_mod = hx.HX()

    hx_mod.plate.thickness = (
        (0.300 + # thickness (in) of coolant plate
         0.375) # thickness (in) of exhaust plate
        * 2.54e-3
        ) # total thickness (m)
    hx_mod.plate.set_h()

    hx_mod.width = 20. * 2.54e-2
    hx_mod.exh.height = 2.5 * 2.54e-2
    hx_mod.cool.height = 1. * 2.54e-2
    hx_mod.length = 20. * 2.54e-2

    hx_mod.te_pair.Ptype.area = (2.e-3) ** 2

    hx_mod.te_pair.leg_area_ratio = 0.662
    hx_mod.te_pair.I = 0.001  # turns off TE effect
    hx_mod.te_pair.length = 1.e-5
    hx_mod.te_pair.fill_fraction = 1.

    hx_mod.te_pair.set_leg_areas()

    hx_mod.te_pair.Ntype.material = 'MgSi'
    hx_mod.te_pair.Ptype.material = 'HMS'

    hx_mod.type = 'counter'

    hx_mod.exh.set_enhancement('IdealFin')
    hx_mod.exh.enh.thickness = 0.1 * 2.54e-2
    hx_mod.exh.enh.spacing = 0.098 * 2.54e-2
    # spacing = 0.200 - 0.124 / 2. - 0.040 = 0.098 in

    hx_mod.cool.set_enhancement('IdealFin')
    hx_mod.cool.enh.thickness = 0.08 * 2.54e-2
    hx_mod.cool.enh.spacing = 0.120 * 2.54e-2
    # spacing = 0.200 - 0.100 / 2. - 0.030 = 0.120 in
```

```
    hx_mod.cool.T_inlet_set = 300.

    return hx_mod

def solve_hx(hx_exp, hx_mod):

    """Solves heat exchanger for all the conditions in the
    experimental data set."""

    hx_mod.Qdot_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.exh.deltaP_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.exh.velocity_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.exh.rho_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.exh.Re_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_exp.exh.Re_omega_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.exh.c_p_bar_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_mod.effectiveness_arr = np.zeros(hx_exp.exh.T_in.size)
    hx_exp.Qdot_omega = np.zeros(hx_exp.exh.T_in.size)
    hx_exp.Qdot_max_omega = np.zeros(hx_exp.exh.T_in.size)
    hx_exp.Qdot_max = np.zeros(hx_exp.exh.T_in.size)
    hx_exp.exh.deltaT_max = hx_exp.exh.T_in - hx_exp.cool.T_out

    hx_exp.exh.T_omega = (
        (0.5 * (hx_exp.exh.T_in + hx_exp.exh.T_out) - 273.15) * 75.e-4
        )

    for i in range(hx_exp.exh.T_in.size):
        hx_mod.exh.T_inlet = hx_exp.exh.T_in[i]
        hx_mod.exh.mdot = hx_exp.exh.mdot[i]
        hx_mod.cool.mdot = (
            hx_exp.cool.Vdot[i] * 3.8e-3 / 60 * hx_mod.cool.rho
            ) # kg / s
        hx_mod.cool.T_outlet = hx_exp.cool.T_out[i]

        hx_mod.solve_hx()

        if hx_exp.exh.T_omega[i] < 2.2:
            hx_exp.exh.T_omega[i] = 2.2

        hx_exp.Qdot_omega[i] = (
            ((hx_mod.exh.c_p_nodes.mean() * hx_exp.exh.deltaT[i] *
        hx_mod.exh.mdot_omega) ** 2. + 2. * (hx_exp.exh.mdot[i] *
        hx_mod.exh.c_p_nodes.mean() * hx_exp.exh.T_omega[i]) ** 2.) **
        0.5
            )
        hx_exp.Qdot_max_omega[i] = (
            ((hx_mod.exh.c_p_nodes.mean() * hx_exp.exh.deltaT[i] *
        hx_mod.exh.mdot_omega) ** 2. + 2. * (hx_exp.exh.mdot[i] *
        hx_mod.exh.c_p_nodes.mean() * hx_exp.exh.T_omega[i]) ** 2.) **
        0.5
            )
        hx_exp.Qdot_max[i] = (
            hx_exp.exh.mdot[i] * hx_mod.exh.c_p_nodes.mean() *
            (hx_exp.exh.T_in - hx_exp.cool.T_in)[i]
            )
        hx_mod.Qdot_arr[i] = hx_mod.Qdot_total
        hx_mod.exh.deltaP_arr[i] = hx_mod.exh.deltaP_total
        hx_mod.exh.velocity_arr[i] = hx_mod.exh.velocity_nodes.mean()
        hx_mod.exh.rho_arr[i] = hx_mod.exh.rho_nodes.mean()
        hx_mod.exh.Re_arr[i] = hx_mod.exh.Re_nodes.mean()
        hx_exp.exh.Re_omega_arr[i] = hx_mod.exh.Re_omega
        hx_mod.exh.c_p_bar_arr[i] = hx_mod.exh.c_p_nodes.mean()
        hx_mod.effectiveness_arr[i] = hx_mod.effectiveness
```

```
    return hx_mod

def fit_hx(hx_exp, hx_mod):
    """Changes parameters in model to make model data fit experimental
    data."""

    from scipy.optimize import leastsq

    def get_fit_error(fit_params, *args):
        hx_mod = args[0]
        hx_mod.R_extra = fit_params
        hx_mod = solve_hx(hx_exp, hx_mod)
        Qdot_err = hx_mod.Qdot_arr - hx_exp.exh.Qdot
        return Qdot_err

    hx_mod.leastsq_out = leastsq(get_fit_error, x0=0, args=hx_mod)
    hx_mod.R_extra = hx_mod.leastsq_out[0]

    return hx_mod
```

E.3 Common Code

All code in this section can be found here:

https://github.com/calbaker/Python/blob/master/properties.py.

Listing E.18: properties.py

```
"""Contains classes for characterizing flows and determining
properties of ideal gases.  It is a good idea to put this script in a
directory that is in the python path."""

import numpy as np
from scipy.integrate import quad

import constants as const

class ideal_gas(object):

    """Class for modeling ideal gases.

    Methods:

    __init__
    get_c_p_air
    get_enthalpy
    get_entropy
    get_mu
    get_n
    get_rho
    set_TempPres_dependents
    set_Temp_dependents
    set_rho
    set_standard_air
    set_thermal_props

    """
```

```
def __init__(self,**kwargs):

    """Sets constants and such.

    Sets the following variables:
    species: name of gas species
    Mhat: molecular weight (kg/kmol) of gas species
    d: collision diameter (m)
    Pr: Prandtl number

    Unless otherwise noted, all gas properties are coming from
    Bird, Stewart, Lightfoot Transport Phenomena Table E.1"""

    if 'species' in kwargs:
        self.species = kwargs['species']
    else:
        self.species = 'air'

    if self.species == 'air':
        self.Mhat = 28.964
        self.d = 3.617e-10
        self.Pr = 0.74

    elif self.species == 'propane' or 'C3H8':
        self.Mhat = 44.10
        self.d = 4.934e-10

    if 'Mhat' in kwargs:
        self.Mhat = kwargs['Mhat']
    if 'd' in kwargs:
        self.d = kwargs['d']
    if 'Pr' in kwargs:
        self.Pr = kwargs['Pr']

    # Calculated attributes
    self.R = const.Rhat / self.Mhat
    # gas constant (kJ/kg*K)
    self.m = self.Mhat / const.Nhat
    # molecular mass (kg/molecule)

def set_standard_air(self):

    """Sets properties of air for T = 300 K and P = 101 kPa.

    Methods:

    self.set_TempPres_dependents

    """

    self.T = 300.
    self.P = 101.
    self.set_TempPres_dependents()

def get_entropy(self,T):

    """Returns entropy with respect to 0 K at 1 bar.

    Inputs:

    T: temperature(K)
```

```
        Returns:

        entropy (kJ / kg / K)

        """

        def get_integrand(T):
            integrand = self.get_c_p_air(T) / T
            return integrand
        entropy = (quad(get_integrand, 0.5, T)[0])
        return entropy

    def get_enthalpy(self,T):

        """Returns enthalpy based on temperature.

        Inputs

        T : temperature (K)

        Returns:

        enthalpy (kJ/kg)"""

        enthalpy = (quad(self.get_c_p_air, 0., T)[0])
        return enthalpy

    def get_rho(self, T, P):

        """Returns density (kg/m^3) as function of T and P

        arguments: P(kPa), T(K)

        returns rho(kg/m**3)

        """

        rho = P / (self.R * T) # density (kg/m**3)
        return rho

    def get_n(self, rho):

        """Returns number density (#/m^3)

        argument: rho(kg/m**3)
        returns n(#/m**3)

        """

        n = rho / self.m # number density (#/m^3)
        return n

    def get_mu(self,T):

        """Returns viscosity (Pa*s) of ideal gas

        Inputs:

        T : temperature (K)

        Returns:

        mu : viscosity (Pa * s)
```

```
        Expression from Bird, Stewart, Lightfoot Eq. 1.4-14.  This
        expression works ok for nonpolar gases, even ones with
        multiple molecules.

        """

        mu = (5. / 16. * (np.pi * self.m * const.k_B * self.T)**0.5 /
        (np.pi * self.d**2))

        return mu

    def get_c_p_air(self,T):

        """c_p (kJ/kg-K) of air

        Calculated using Moran and Shapiro, Table A-21 constants for
        polynomial for specific heat of air

        Inputs:

        T : temperature (K)

        Returns:

        c_p_air : specific heat (kJ / (kg * K)) for air

        """

        self.polyrep = np.poly1d([0.2763e-12, 1.913e-9, 3.294e-6,
        -1.337e-3, 3.653])
        c_p_air = self.polyrep(T) * self.R

        return c_p_air

    def set_rho(self):

        """Sets mass density and number density

        rho (kg/m^3)
        n (#/m^3)

        """

        self.rho = self.get_rho(self.T, self.P)
        self.n = self.get_n(self.rho)

    def set_Temp_dependents(self):

        """Sets several properties that are temp dependent.

        Sets viscosity (Pa*s) of general ideal gas and specific
        heat (kJ/kg*K) of air."""

        self.mu = self.get_mu(self.T)
        self.c_p_air = self.get_c_p_air(self.T)
        # constant pressure specific heat of air (kJ/kg*K)
        if np.isscalar(self.T) == True:
            self.entropy = self.get_entropy(self.T)
            self.enthalpy = self.get_enthalpy(self.T)
        elif self.T.size == 1:
            self.entropy = self.get_entropy(self.T)
            self.enthalpy = self.get_enthalpy(self.T)
```

```
def set_TempPres_dependents(self):
    """Sets temperature and pressure dependent properties.

    Properties include density (kg/m^3), kinematic viscosity
    (m^2/s), thermal diffusivity (m^2/s), and thermal conductivity
    (kW/m-K)

    Methods:

    self.set_Temp_dependents
    self.set_rho

    """

    self.set_Temp_dependents()
    self.set_rho()
    self.nu = self.mu / self.rho # kinematic viscosity (m^2/s)

def set_thermal_props(self):

    """Sets a bunch of properties.

    Sets temp and press dependents, then sets thermal
    diffusivity (m^2/s) if Pr is known, and then sets thermal
    conductivity (kW/m*K).

    Methods:

    self.set_TempPres_dependents

    """

    self.set_TempPres_dependents()
    self.alpha = self.nu / self.Pr # thermal diffusivity (m^2/s)
    self.k_air = (self.alpha * self.rho * self.c_p_air) # thermal
        # conductivity(kW/m-K) of air
```

Bibliography

[1] J. G. Calvert, J. B. Heywood, R. F. Sawyer, and J. H. Seinfeld. Achieving acceptable air quality: some reflections on controlling vehicle emissions. *Science*, 261(5117):37–45, 1993.

[2] Willard W. Pulkrabek. *Engineering Fundamentals of the Internal Combustion Engine*. Prentice Hall, 2 edition, June 2003.

[3] United States. Environmental Protection Agency. Office of Air. *Smog, who does it hurt?* US Environmental Protection Agency, Air and Radiation, 1999.

[4] OAR US EPA. Ozone good up high bad nearby - why bad nearby. http://www.epa.gov/airquality/gooduphigh/bad.html#6. Ozone Good Up High Bad Nearby is a brochure explaining why ozone is good for the environment when present in the upper atmosphere (the stratosphere) and why ozone is bad when present in the lower atmosphere where we can breathe it.

[5] Victor B Flatt. Gasping for breath: The administrative flaws of federal hazardous air pollution regulation and what we can learn from the states. *Ecology Law Quarterly*, 34:107, 2007.

[6] G. D'Amato. Effects of climatic changes and urban air pollution on the rising trends of respiratory allergy and asthma. *Multidisciplinary Respiratory Medicine*, 6(1):28, 2011.

[7] Jun Kagawa. Health effects of diesel exhaust emissions—a mixture of air pollutants of worldwide concern. *Toxicology*, 181–182(0):349–353, December 2002.

[8] Günter Oberdörster. Pulmonary effects of inhaled ultrafine particles. *International Archives of Occupational and Environmental Health*, 74(1):1–8, 2000.

[9] OAR US EPA. Second prospective study - 1990 to 2020 | benefits and costs of the clean air act | US EPA. http://www.epa.gov/air/sect812/prospective2.html.

[10] Ronald M Heck, Robert J Farrauto, and Suresh T Gulati. *Catalytic Air Pollution Control: Commercial Technology*. Wiley, 3 edition, February 2009.

[11] P. Yang, H. Yan, S. Mao, R. Russo, J. Johnson, R. Saykally, N. Morris, J. Pham, R. He, and H. J Choi. Controlled growth of ZnO nanowires and their optical properties. *Advanced Functional Materials*, 12(5):323, 2002.

[12] L. E Greene, M. Law, J. Goldberger, F. Kim, J. C Johnson, Y. Zhang, R. J Saykally, and P. Yang. Low-temperature wafer-scale production of ZnO nanowire arrays. *Angewandte Chemie International Edition*, 42(26):3031–3034, 2003.

[13] Lori E. Greene, Matt Law, Dawud H. Tan, Max Montano, Josh Goldberger, Gabor Somorjai, and Peidong Yang. General route to vertical ZnO nanowire arrays using textured ZnO seeds. *Nano Letters*, 5(7):1231–1236, July 2005.

[14] Lori E Greene, Benjamin D Yuhas, Matt Law, David Zitoun, and Peidong Yang. Solution-grown zinc oxide nanowires. *Inorganic Chemistry*, 45(19):7535–7543, September 2006.

[15] Xudong Wang, Jinhui Song, and Zhong Lin Wang. Nanowire and nanobelt arrays of zinc oxide from synthesis to properties and to novel devices. *Journal of Materials Chemistry*, 17(8):711, 2007.

[16] Yiying Wu, Haoquan Yan, Michael Huang, Benjamin Messer, Jae Hee Song, and Peidong Yang. Inorganic semiconductor nanowires: Rational growth, assembly,

and novel properties. *Chemistry - A European Journal*, 8(6):1260–1268, March 2002.

[17] Xiao-Ning Guo, Ru-Jing Shang, Dong-Hua Wang, Guo-Qiang Jin, Xiang-Yun Guo, and K. N. Tu. Avoiding loss of catalytic activity of pd nanoparticles partially embedded in nanoditches in SiC nanowires. *Nanoscale Research Letters*, 5(2):332–337, November 2009.

[18] G. Wang, E. Johannessen, C. R Kleijn, S. W de Leeuw, and M. O Coppens. Optimizing transport in nanostructured catalysts: a computational study. *Chemical Engineering Science*, 62(18-20):5110–5116, 2007.

[19] Kourosh Malek and Marc-Olivier Coppens. Knudsen self- and fickian diffusion in rough nanoporous media. *The Journal of Chemical Physics*, 119(5):2801, 2003.

[20] Marc-Olivier Coppens. Characterization of fractal surface roughness and its influence on diffusion and reaction. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 187-188:257–265, August 2001.

[21] M.O. Coppens. The effect of fractal surface roughness on diffusion and reaction in porous catalysts-from fundamentals to practical applications. *Catalysis Today*, 53(2):225–243, 1999.

[22] R. Naumann d'Alnoncourt, X. Xia, J. Strunk, E. Löffler, O. Hinrichsen, and M. Muhler. The influence of strongly reducing conditions on strong metal–support interactions in Cu/ZnO catalysts used for methanol synthesis. *Physical Chemistry Chemical Physics*, 8(13):1525, 2006.

[23] Paul M. Laing, Michael D. Shane, Seha Son, Andrew A. Adamczyk, and Peter Li. A simplified approach to modeling exhaust system emissions: SIMTWC.

Technical Report 1999-01-3476, SAE International, Warrendale, PA, October 1999.

[24] GC Koltsakis, PA Konstantinidis, and AM Stamatelos. Development and application range of mathematical models for 3-way catalytic converters. *Applied Catalysis B: Environmental*, 12(2-3):161–191, 1997.

[25] Christine K. Lambert, Paul M. Laing, and Robert H. Hammerle. Using diesel aftertreatment models to guide system design for tier II emission standards. Technical Report 2002-01-1868, SAE International, Warrendale, PA, June 2002.

[26] Santhoji Katare and Paul M. Laing. A hybrid framework for modeling aftertreatment systems: A diesel oxidation catalyst application. Technical Report 2006-01-0689, SAE International, Warrendale, PA, April 2006.

[27] Santhoji Katare, James M. Caruthers, W. Nicholas Delgass, and Venkat Venkatasubramanian. An intelligent system for reaction kinetic modeling and catalyst design. *Industrial & Engineering Chemistry Research*, 43(14):3484–3512, July 2004.

[28] Madhuchhanda Bhattacharya, Michael P. Harold, and Vemuri Balakotaiah. Shape normalization for catalytic monoliths. *Chemical Engineering Science*, 59(18):3737–3766, September 2004.

[29] Madhuchhanda Bhattacharya, Michael P. Harold, and Vemuri Balakotaiah. Low-dimensional models for homogeneous stirred tank reactors. *Chemical Engineering Science*, 59(22-23):5587–5596, November 2004.

[30] Saurabh Y. Joshi, Michael P. Harold, and Vemuri Balakotaiah. Low-dimensional

models for real time simulations of catalytic monoliths. *AIChE Journal*, 55(7):1771–1783, July 2009.

[31] Saurabh Y. Joshi, Michael P. Harold, and Vemuri Balakotaiah. On the use of internal mass transfer coefficients in modeling of diffusion and reaction in catalytic monoliths. *Chemical Engineering Science*, 64(23):4976–4991, December 2009.

[32] Saurabh Y. Joshi, Michael P. Harold, and Vemuri Balakotaiah. Overall mass transfer coefficients and controlling regimes in catalytic monoliths. *Chemical Engineering Science*, 65(5):1729–1747, March 2010.

[33] M. Aryafar and F. Zaera. Kinetic study of the catalytic oxidation of alkanes over nickel, palladium, and platinum foils. *Catalysis Letters*, 48(3):173–183, 1997.

[34] Chad Allan Baker. *Vapor Transport Techniques for Growing Macroscopically Uniform Zinc Oxide Nanowires*. PhD thesis, University of Texas Libraries, Austin, Tex, August 2009.

[35] Ted pella sputter coaters, carbon coaters, vacuum evaporator. http://www.tedpella.com/cressing_html/intro.htm.

[36] J Miller. A fundamental study of platinum tetraammine impregnation of silica2. the effect of method of preparation, loading, and calcination temperature on (reduced) particle size. *Journal of Catalysis*, 225(1):203–212, July 2004.

[37] AA Kolmakov and D. W. Goodman. Size effects in catalysis by supported metal clusters. In S. Khanna and W. Castleman, editors, *Quantum Phenomena in Clusters and Nanostructures*, pages 159–197. Springer: Berlin Heidelberg, 2003.

[38] L. Pisani. Multi-component gas mixture diffusion through porous media: A 1D analytical solution. *International Journal of Heat and Mass Transfer*, 51(3-4):650–660, 2008.

[39] Massimo Morbidelli, Asterios Gavriilidis, and Arvind Varma. *Catalyst design: optimal distribution of catalyst in pellets, reactors, and membranes*. Cambridge University Press, February 2001.

[40] A. F Mills. *Heat Transfer*. Prentice Hall, 2 edition, August 1998.

[41] R. Byron Bird, Warren E Stewart, and Edwin N Lightfoot. *Transport Phenomena, 2nd Edition*. Wiley, 2 edition, July 2001.

[42] F. Gao, S.M. McClure, Y. Cai, K.K. Gath, Y. Wang, M.S. Chen, Q.L. Guo, and D.W. Goodman. CO oxidation trends on pt-group metals from ultrahigh vacuum to near atmospheric pressures: A combined in situ PM-IRAS and reaction kinetics study. *Surface Science*, 603(1):65–70, January 2009.

[43] Adrian Bejan. Optimal internal structure of volumes cooled by single-phase forced and natural convection. *Journal of Electronic Packaging*, 125(2):200, 2003.

[44] A.K. da Silva, S. Lorente, and A. Bejan. Constructal multi-scale structures for maximal heat transfer density. *Energy*, 31(5):620–635, April 2006.

[45] A. K. da Silva and A. Bejan. Constructal multi-scale structure for maximal heat transfer density in natural convection. *International journal of heat and fluid flow*, 26(1):34–44, 2005.

[46] A. K. da Silva, A. Bejan, and S. Lorente. MAXIMAL HEAT TRANSFER DENSITY IN VERTICAL MORPHING CHANNELS WITH NATURAL CONVEC-

TION. *Numerical Heat Transfer, Part A: Applications*, 45(2):135–152, January 2004.

[47] John Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill Science/Engineering/Math, 1 edition, April 1988.

[48] J. A Caton. Operating characteristics of a spark-ignition engine using the second law of thermodynamics: effects of speed and load. *SAE Technical Paper*, page 2000–01–0952, 2000.

[49] T. Endo, S. Kawajiri, Y. Kojima, K. Takahashi, T. Baba, S. Ibaraki, T. Takahashi, and M. Shinohara. Study on maximizing exergy in automotive engines. Technical Report 2007-01-0257, SAE International, Warrendale, PA, April 2007.

[50] Erik W Miller, Terry J Hendricks, and Richard B Peterson. Modeling energy recovery using thermoelectric conversion integrated with an organic rankine bottoming cycle. *Journal of Electronic Materials*, 38(7):1206–1213, March 2009.

[51] W.M.S.R. Weerasinghe, R.K. Stobart, and S.M. Hounsham. Thermal efficiency improvement in high output diesel engines a comparison of a rankine cycle with turbo-compounding. *Applied Thermal Engineering*, 30(14–15):2253–2256, October 2010.

[52] Hugues L. Talom and Asfaw Beyene. Heat recovery from automotive engine. *Applied Thermal Engineering*, 29(2–3):439–444, February 2009.

[53] Terry J Hendricks. Thermal system interactions in optimizing advanced thermoelectric energy recovery systems. *Journal of Energy Resources Technology*, 129(3):223–231, 2007.

[54] T. J Hendricks and J. A Lustbader. Advanced thermoelectric power system investigations for light-duty and heavy duty applications. II. In *Thermoelectrics*, page 387– 394. IEEE, August 2002.

[55] Quazi E Hussain, David R Brigham, and Clay W Maranville. Thermoelectric exhaust heat recovery for hybrid vehicles. Technical Report 2009-01-1327, SAE International, Warrendale, PA, April 2009.

[56] Doug T. Crane. An introduction to system level steady-state and transient modeling and optimization of HighPower density thermoelectric generator devices made of segmented thermoelectric elements. *Journal of Electronic Materials*, 40(5):561–569, 2011.

[57] Douglas T. Crane and Gregory S. Jackson. Optimization of cross flow heat exchangers for thermoelectric waste heat recovery. *Energy Conversion and Management*, 45(9–10):1565–1582, June 2004.

[58] Douglas Crane, Gregory Jackson, and David Holloway. Towards optimization of automotive waste heat recovery using thermoelectrics. Technical Report 2001-01-1021, SAE International, Warrendale, PA, March 2001.

[59] David Michael Rowe. *Thermoelectrics Handbook: Macro to Nano*. CRC/Taylor & Francis, Boca Raton, 2006.

[60] Richard Stobart and Dan Milner. The potential for thermo-electric regeneration of energy in vehicles. Technical Report 2009-01-1333, SAE International, Warrendale, PA, April 2009.

[61] Richard K Stobart, Anusha Wijewardane, and Chris Allen. The potential for

thermo-electric devices in passenger vehicle applications. Technical Report 2010-01-0833, SAE International, Warrendale, PA, April 2010.

[62] Gang Chen. *Nanoscale Energy Transport and Conversion: A Parallel Treatment of Electrons, Molecules, Phonons, and Photons*. Oxford University Press, USA, March 2005.

[63] Charles A. Domenicali. Irreversible thermodynamics of thermoelectric effects in inhomogeneous, anisotropic media. *Physical Review*, 92(4):877–881, November 1953.

[64] T. P Hogan and T. Shih. *Modeling and characterization of power generation modules based on bulk materials*. Boca Raton, FL: CRC Press, 2006. Eq. 12.25 contains a typo. Equation 10 in this paper is the correct form.

[65] Adrian Bejan. *Convection Heat Transfer*. Wiley, 3 edition, July 2004.

[66] Frank P Incropera, David P DeWitt, Theodore L Bergman, and Adrienne S Lavine. *Fundamentals of Heat and Mass Transfer*. Wiley, 5th edition, April 2011.

[67] Y. Hori, D. Kusano, T. Ito, and K. Izumi. Analysis on thermo-mechanical stress of thermoelectric module. In *Eighteenth International Conference on Thermoelectrics, 1999*, pages 328 –331, September 1999.

[68] Bryan C. Gundrum, David G. Cahill, and Robert S. Averback. Thermal conductance of metal-metal interfaces. *Physical Review B*, 72(24):245426, December 2005.

[69] Hongli Gao, Tiejun Zhu, Xinxin Liu, Luxin Chen, and Xinbing Zhao. Flux synthesis and thermoelectric properties of eco-friendly sb doped Mg2Si0.5Sn0.5 solid

solutions for energy harvesting. *Journal of Materials Chemistry*, 21(16):5933, 2011.

[70] W. H. Luo, H. Li, Y. G. Yan, Z. B. Lin, X. F. Tang, Q. J. Zhang, and C. Uher. Rapid synthesis of high thermoelectric performance higher manganese silicide with in-situ formed nano-phase of MnSi. *INTERMETALLICS*, 19(3):404–408, March 2011.

[71] Daniel Kraemer, Bed Poudel, Hsien-Ping Feng, J. Christopher Caylor, Bo Yu, Xiao Yan, Yi Ma, Xiaowei Wang, Dezhi Wang, Andrew Muto, Kenneth McEnaney, Matteo Chiesa, Zhifeng Ren, and Gang Chen. High-performance flat-panel solar thermoelectric generators with high thermal concentration. *Nature Materials*, May 2011.

[72] Chad Baker, Prem Vuppuluri, Li Shi, and Matthew Hall. Model of heat exchangers for waste heat recovery from diesel engine exhaust for thermoelectric power generation. *Journal of Electronic Materials*, February 2012.

[73] J.A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308, 1965.

[74] S. Mancin, C. Zilio, A. Cavallini, and L. Rossetto. Pressure drop during air flow in aluminum foams. *International Journal of Heat and Mass Transfer*, 53(15-16):3121–3130, 2010.

[75] ERG: duocel® - aluminum foam properties. http://www.ergaerospace.com/Aluminum-properties.htm.

[76] Aaron M. Huber and Raymond Viskanta. Effect of jet-jet spacing on convective

heat transfer to confined, impinging arrays of axisymmetric air jets. *International Journal of Heat and Mass Transfer*, 37(18):2859–2869, December 1994.

[77] Robert W. Fox, Alan T. McDonald, and Philip J. Pritchard. *Introduction to Fluid Mechanics*. John Wiley & Sons, 6th edition, May 2005.

[78] H. S. Lee. *Thermal Design: Heat Sinks, Thermoelectrics, Heat Pipes, Compact Heat Exchangers, and Solar Cells*. John Wiley and Sons, November 2010.

[79] Raj M. Manglik and Arthur E. Bergles. Heat transfer and pressure drop correlations for the rectangular offset strip fin compact heat exchanger. *Experimental Thermal and Fluid Science*, 10(2):171–180, February 1995.

[80] W. M. Kays and A. L. London. *Compact Heat Exchangers*. Krieger Pub Co, 3 sub edition, January 1998.

[81] Extruded aluminum heatsinks from HeatSinkUSA. http://www.heatsinkusa.com/.

[82] Chen Li and G.P. Peterson. The effective thermal conductivity of wire screen. *International Journal of Heat and Mass Transfer*, 49(21-22):4095–4105, October 2006.

[83] Michael J. Moran and Howard N. Shapiro. *Fundamentals of Engineering Thermodynamics*. Wiley, 5th edition, June 2003.

[84] J.P. Holman. *Experimental Methods for Engineers*. McGraw-Hill, seventh edition edition, 2001.

[85] Timothy Thomas Diller. *Development, characterization, and modeling of an electronic particulate matter sensor for internal combustion engines*. PhD thesis, University of Texas, [Austin, Tex, 2009.